THESIS ON INFORMATICS AND SYSTEM ENGINEERING C44

Security Protocols Analysis in the Computational Model — Dependency Flow Graphs-Based Approach

ILJA TŠAHHIROV

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Informatics

Dissertation was accepted for the defence of the degree of Doctor of Philosophy in Engineering on November 12, 2008.

Defence of the thesis: December 15, 2008

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

/ Ilja Tšahhirov /

ISSN 1406-4731
ISBN 978-9985-59-869-6

INFORMAATIKA JA SÜSTEEMITEHNIKA C44

Turvaprotokollide analüüs arvutuslikul mudelil — sõltuvusgraafidel põhinev lähenemisviis

ILJA TŠAHHIROV

Acknowledgements

I would like to thank my supervisor Professor Jaak Tepandi for introducing me to the topic of cryptographic systems interoperability, which served as a solid basis for my research, for accepting my decision to adjust the initial direction of the research into the area of the cryptographic protocols analysis, and for all the support and advice during the years I spent working on this thesis.

I am thankful to Peeter Laud for introducing me to the world of language-based security, the ideas, support and guidance he provided, essentially, fulfilling the role of my second supervisor, and a pleasant experience we had while working on the theory and analyser prototype.

I also wish to thank the managers of the Department of Informatics of the Tallinn University of Technology, who provided me with a great support in dealing with formal matters, which allowed me to completely concentrate on the research.

I would like to thank the Doctoral School in ICT (Measure 1.1. of the Estonian NDP for the Implementation of the European SF) and the Estonian Information Technology Foundation for the financial support of this research.

Finally, I thank my parents and wife Ksenia for their support and patience.

Contents

1 Introduction

Security is an important aspect of a computer system. Information security, as defined in [36], means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction.

The security comprises the concepts of confidentiality, integrity, and availability:

> **Confidentiality** is preventing the disclosure of information to the unauthorized systems or individuals;
>
> **Integrity** is preventing an unauthorized modification of information;
>
> **Availability** means that the system services are available when needed.

Most of modern computer systems — ranging from a simple client-server applications on a mobile phone to complex clustered systems performing a distributed computation — consist of several computing entities, connected via network. An important aspect of such systems is the communication protocol — a convention that enables the connection, communication, and data exchange between several computing entities, or participants.

The design of a protocol has a significant impact on the security properties of the system based on it — the flaw in the protocol can make the system vulnerable to the security attacks, even if the individual system components operate properly. Therefore there is a need for methods for indicating whether the given protocol satisfies the given security property or not. The latter is simpler than the first — having an example of successful attack breaching the property of interest is sufficient to show that the protocol does not satisfy it. Showing that the protocol is secure with respect to certain properties is essentially convincing ourselves that no attack (known or unknown) breaching these properties exists.

The topic of this thesis is the development of such method. We focus on confidentiality and integrity properties. The system availability is also important (the operation of any reasonable system should be hard to disrupt), but is different from the first two properties, as both confidentiality and integrity mean the absence of undesired functionality, while the availability means that the system desired functionality is present. Therefore we advocate that the single theory can be well-suited for proving either the confidentiality and integrity, or the availability. In our analysis we concentrate on the first two properties.

Each security analysis is based on certain assumptions about the adversary. The adversary model we use is an *active adversary*. It is quite powerful model — we assume that the whole network and the scheduling of participants execution is fully controlled by the adversary. We advocate that while considering the design of the protocols it is the most reasonable model, as in most of the real life setups the network contains fragments, which are under the control of a third party, therefore there's no guarantee that the communication is not being listened to or tampered with. Also, if the security property holds in this model, it also holds with less powerful (e.g. passive) adversary.

1.1 Dependency analysis

As the communication protocol is one of the key system design components, replacing it on already developed system is extremely hard and is usually associated with high costs (as the system needs to be essentially redeveloped). Therefore, for the system designer it makes sense to analyze the security properties of the protocol *before* it is actually implemented and used. This type of analysis is similar to program static analysis.

The topic of this thesis is dependency analysis. We analyze the dependencies between the computations and values sent to or received from the network in order to see whether the security properties hold.

If the value sent to the network does not depend on a secret data, we conclude that there is no information flow from the secret data to this public output, and, therefore, the confidentiality of the protocol is not broken. If this statement holds for all the values which can be sent to the network, we conclude that the confidentiality property holds for a given protocol.

By tracking the dependencies between the operations, it is also possible to examine whether the certain invariants hold in any run of the protocol. It is possible to represent the system integrity properties by choosing these invariants, and then, by proving these invariants, prove that the integrity property of interest holds.

Another key concepts of our approach are protocol indistinguishability and transformations. Two protocols are indistinguishable if by observing their public outputs (in case of active adversary, also tampering with them) it is still not possible to determine (with non-negligible probability) which of the two protocols is run. If the protocols are indistinguishable, then they are both secure or insecure. Therefore, instead of analyzing a protocol, it is possible to analyze another protocol, indistinguishable from the original one, and get the same results. Using the approach, we get a sequence of games, where the adversary advantage decreases only negligible from game to game, and in the final game the adversary has obviously no advantage at all.

In this thesis we propose a framework for examining the protocol security properties based on the notion of computational indistinguishability, assuming certain properties of cryptographic operations. Currently the framework is used for checking the preservation of confidentiality and integrity properties in protocols. In principle, the framework should be suitable for verifying all properties whose fulfilment can be observed by the protocol participants and/or the adversary — the transformation process does not change the observable properties of protocols.

1.2 Use of Cryptography

In order to prevent unauthorized access to the information and / or tampering with it, the security-related functions and cryptographic methods are used. Each such function or method should have certain properties — for instance it should be hard to decrypt the encrypted information without having the corresponding decryption key. The notion of "hardness" can be formulated (on the example of decryption) using one of the following approaches:

> The formal approach, also referred to as a Dolev-Yao model [24], assuming that without having a key it is not possible to decrypt the message at all.
>
> The complexity-theory based approach [35], based on a more realistic assumption that the decryption not having the key is possible, but with a very small probability (a function of the length of the key).

Our analysis relies on the complexity-theoretic definitions. The result of the protocol security examination is stated in form: the protocol is secure (except for the cases, which can occur only with a negligible probability), assuming that the cryptographic methods, used in it, are secure (also, in the complexity-theoretic sense).

Any cryptographic primitive, the properties of which can be formalized this way, can be used in the protocol, serving as the input for our analysis. In this thesis we supply the definitions for the symmetric (secret key) encryption, asymmetric (public key) encryption, and digital signature.

1.3 Related Work and Our Contribution

The research presented in this thesis belongs to a body of work attempting to bridge the gap between the two main approaches for modeling and analyzing the cryptographic protocols — the Dolev-Yao model [24] and the complexity-theory-based approach [35]. The most related, and also a source of inspiration to our work has been the protocol analysis framework by Blanchet [18, 19, 20]. The difference of our approaches is in the choice of the program (protocol) representation. We believe that the use of dependency graphs instead of a more conventional representation allows our analyzer to better focus on the important details of data and control flow. Both frameworks are based on the view of cryptographic proofs as sequences of games [17, 10]. In both frameworks, in order to prove a protocol correct, the automated analyzer constructs such a sequence where the adversary's advantage diminishes only negligibly from one game to the next one, and where the adversary has obviously no advantage in the final game. Another difference with [18, 19, 20] is the degree of automation – the analyzer from [18, 19, 20] still requires the human-produced hints on the set and order of transformations to apply for some protocols; the analyzer prototype implemented based on our framework does not require any hints.

The work in this area has been started by Abadi and Rogaway [13], who considered the relationship of formal and computational symmetric encryption under passive attacks and provided a procedure to check whether two formal messages have indistinguishable computational semantics. The same primitive and class of attacks have been further considered in [14, 31, 26, 30, 15], in these papers the language has been expanded (the constraints have been weakened) and the security definitions have been clarified. Further on, the active attacks with the range of cryptographic primitives were considered in [22, 25, 29] (based on the translation of protocol traces from the computational to the formal model) and [27, 16] (based on the application of the universally composable [21, 32] cryptographic library [4]).

Another body of research the present work is based on is the static analysis on the intermediate program representations. We use the protocol representation close to [33]. Unlike some of the frameworks based on protocol rewriting – [10, 12], the protocol transformations we perform do not produce several sub-protocols which are to be analyzed separately; the representation chosen is capable of holding all the possible information flows and execution variants, therefore having better potential for analyzing the replication and data flows between different protocol runs.

The basis of the analysis — protocol semantics and the set of assumptions on the cryptographic operations we used — is similar to [10, 12], while the protocol language is much more powerful, and the analysis methodology is significantly different. The basic principles of applying the dependency graph for this purpose are outlined in [34]. The integrity properties, which can be further investigated using the developed approach range from traditional definitions of protocol integrity (injective and non-injective agreement) to system interoperability (e.g. to formally prove the findings described in [11]).

Our main contribution, distinguishing our work from the works mentioned above, is the introduction of the support for the replication to the analyzed protocols, and using the approach based on the extensive usage of the dependency graphs, similar to those introduced in [33], for the analysis of the protocols' security. Also, the framework described in this thesis, is suitable for producing the automated, computationally sound proofs for the protocols security.

1.4 Thesis Overview

This thesis has the following structure.

The terms, abbreviations, and symbols, used in the thesis, are described in the sec. 2. Then, in sec. 3, we describe the procedural language for specifying the input to our analysis: both the syntax (sec. 3.1) and the execution semantics (sec. 3.2) are covered. The language itself is similar to [37], but contains the means for specifying the computations performed during the protocol execution with enough details to perform the analysis.

Sec. 4 is dedicated to the dependency graphs — the protocol representation used for the analysis. First, the graphs of a finite size are described (sec. 4.1), followed by more complex case of a graph with infinite number of nodes (sec. 4.2). The execution semantics of the dependency graphs are defined in sec. 4.3. The section is concluded by defining the indistinguishability of two protocols, represented by the dependency graphs (sec. 4.4)

The method for translating a protocol (preserving its semantics) from the procedural language to the dependency graph language is described in sec. 5. The language for specifying the dependency graph transformations is described in sec. 6, followed by the transformations we used, specified in this language (sec. 7).

The final sections of the thesis discuss the achieved practical results (sec. 8), conclusions, and possible directions for future research (sec. 9).

The definition of the properties of the cryptographic operations we used (this is just one possible definition, not being the part of the core framework, as different set of cryptographic primitives and/or stronger/weaker properties can be considered as well), and the full list of transformations we defined, reside in the thesis appendix.

2 Abbreviations, Terms, and Symbols

The terms and symbols used in the thesis are listed in table 1.

Table 1: Terms and Symbols

Symbol	Explanation
DG	Dependency Graph
DGR	Dependency Graph Representation
DGF	Dependency Graph Fragment
DGFR	Dependency Graph Fragment Representation
$\mathbb{B}$	Set of boolean values (true and false)
$\mathbf{N}$	Set of numeric values
$\mathcal{P}$	Power set
$\mathbf{Dim}$, d	A category of replication dimensions and its member
$\mathbf{Num}$, n	A category of numerals and its member
$\mathbf{Aexpr}$, a	A category of arithmetic expressions and its member
$\mathbf{Var}$, x	A category of variables and its member
$\mathbf{Bexpr}$, b	A category of boolean expressions and its member
$\mathbf{Expr}$, e	A category of expressions and its member
$\mathbf{Stmt}$, S	A category of protocol statements and its member
$\mathbf{Lab}$, l	A category of protocol expression labels and its member
$\mathcal{A}$	Arithmetic expression semantics function
$\mathcal{B}$	Boolean expression semantics function
$\mathcal{S}$	Protocol statement semantics function
Lab	Set of DG/DGR labels
$Lab_{\rightarrow}$	Set of DG/DGR labels of Send-nodes
$Lab_{\leftarrow}$	Set of DG/DGR labels of Receive and Req-nodes
Lab_{*}	Set of DG/DGR labels of RS and Secret-nodes
Lab^{f}	Set of DGF/DGFR labels
$Lab^{f}{}_{\rightarrow}$	Set of DGF/DGFR labels of OutputB and OutputS-nodes
$Lab^{f}{}_{\leftarrow}$	Set of DGF/DGFR labels of InputB and InputS-nodes
$Lab^{f}{}_{*}$	Set of DGF/DGFR labels of RS and Secret-nodes
$\mathbf{GR}$, gr	A category of DGRs and its member
$\mathbf{G}$, g	A category of DGs and its member
Fix	Fixed point operator

3 Protocol Language

We represent a protocol, which is the input to our analysis, in a WHILE-style procedural language. The language comprises statements, which, in turn, may contain arithmetic or boolean expressions. So-called atomic statement represents a single operation (in our case — variable assignment, sending the value to the network, and the instructions for marking the initiation and completion of the session (needed to investigate the integrity property). The atomic statements can be composed into the compositional statements (or just statements). Using the compositional statements we define the sequential, parallel, or conditional execution of the atomic statements. Process replication is also supported. The common expressions are supported: constant generation, tupling, projection, boolean logic operations, and so on. We've supplied the definition of several cryptographic operations as examples (asymmetric encryption, symmetric encryption, nonce generation, and digital signature); more cryptographic operations can be brought in if needed for the purpose of analyzing a particular protocol. The language is formally described in the remained of this section.

3.1 Syntactic Definition

The language consists of the following syntactic categories. For each category, the name of the meta-variable (which can be sub- or superscripted), ranging over it, is given.

- n ranges over numerals, **Num**;
- x ranges over variables, **Var**;
- l ranges over expression and statement labels, **Lab**; the label of the statement or expression is used to identify the "external" values consumed or produced by the statement or expression. The examples are: value sent to or received from the network, and random coins;
- d ranges over replication dimension names, **Dim**; Each protocol statement can be replicated (executed many times in parallel). If the statement is replicated, it has the coordinate in (exactly one) replication dimension. The value of the coordinate is used to identify the particular instance of the replicate operation;
- a ranges over arithmetic expressions, **Aexpr**;
- b ranges over boolean expressions, **Bexpr**;

- AS ranges over atomic statements (each of this statements can be executed in one step), **AStmt**;
- S ranges over compositional statements, **Stmt**.

The exact structure of numerals, variables, labels, and dimension names is not defined here. Any reasonable implementation (for instance, numbers being sequence of digits and other mentioned categories being the sequence of letters) would be be sufficient for our purposes, so we leave it flexible. The remaining syntactic categories are defined in Figure 1. Given the above definitions, the

$$
\begin{array}{lclclcl}
a & ::= & x & \mid & n & \mid & \mathit{nonce}^l \\
 & \mid & \mathit{keypair}^l & \mid & \mathit{pubkey}(a) & \mid & \mathit{pubenc}^l(a_1, a_2) \\
 & \mid & \mathit{pubdec}(a_1, a_2) & \mid & \mathit{svkpair}^l & \mid & \mathit{verkey}(a) \\
 & \mid & \mathit{sig}^l(a_1, a_2) & \mid & \mathit{getmsg}(a) & \mid & \mathit{symkey}^l \\
 & \mid & \mathit{symenc}^l(a_1, a_2) & \mid & \mathit{symdec}(a_1, a_2) & \mid & \mathit{receive}^l \\
 & \mid & \mathit{tuple}_n(a_1, \ldots, a_n) & \mid & \mathit{proj}_{n,n'}(a) & \mid & \mathit{secret}^l \\
b & ::= & a_1 = a_2 & \mid & a_1 \neq a_2 & \mid & \mathit{testsig}(a_1, a_2) \\
 & \mid & \mathit{and}(b_1, b_2) & \mid & \mathit{or}(b_1, b_2) & \mid & \mathit{isok}(a) \\
AS & ::= & x := a & \mid & \mathit{send}^l\, a & \mid & \mathit{begin}^l\, a \\
 & \mid & \mathit{end}^l\, a & & & & \\
S & ::= & \mathit{par}_n(S_1, \ldots, S_n) & \mid & \mathit{rep}(d, S) & \mid & \mathit{if}\ b\ \mathit{then}\ S_1 \\
 & \mid & \mathit{iff}\ b\ \mathit{then}\ AS\ \mathit{finally}\ S & \mid & \mathit{stop} & \mid & AS; S
\end{array}
$$

Figure 1: Syntactic Categories of the Protocol Language

protocol is described by $S \in \mathbf{Stmt}$.

To illustrate the capabilities of the language, let us consider the well-known Needham-Schroeder-Lowe key-exchange protocol:

$$
\begin{array}{rl}
A \to B : & \{N_A, A\}_{\mathrm{pk}(B)} \\
B \to A : & \{N_A, N_B, B\}_{\mathrm{pk}(A)} \\
A \to B : & \{N_B\}_{\mathrm{pk}(B)}
\end{array}
$$

This public-key protocol is used to establish the shared secrets N_A and N_B between A and B. In our modeling, N_B is the shared key K_{AB} established between A and B. To examine the secrecy of payloads, we add an extra message from B to A containing the encryption of a secret payload under the key K_{AB}. Hence the protocol we're currently considering is

$$\begin{array}{rl} A \to B: & \{N_A, A\}_{\mathrm{pk}(B)} \\ B \to A: & \{N_A, K_{AB}, B\}_{\mathrm{pk}(A)} \\ A \to B: & \{K_{AB}\}_{\mathrm{pk}(B)} \\ B \to A: & \{M\}_{K_{AB}} \\ A \to \quad : & \mathrm{OK} \end{array} \tag{1}$$

I.e. the party A acknowledges the receipt of the secret M after the received message has been successfully decrypted.

This protocol is depicted by the following protocol language statement:

```
// Protocol Initialization
   $\mathrm{sk}(A) := \mathit{keypair}^1;$
   $\mathrm{pk}(A) := \mathit{pubkey};$
   $\mathrm{sk}(B) := \mathit{keypair}^2;$
   $\mathrm{pk}(B) := \mathit{pubkey};$
   $\mathit{par}_2($
// Participant A
   $N_A := \mathit{nonce}^3;$
   $\mathit{send}^4\ \mathit{pubenc}^5(\mathrm{pk}(B), \mathit{tuple}(N_A, \mathrm{pk}(A)));$
   $m_2 := \mathit{pubdec}(\mathrm{sk}(A), \mathit{receive}^{10});$
   $\mathit{if}\ \mathit{proj}_{1,3}(m_2) = N_A\ \mathit{then}$
   $\mathit{if}\ \mathit{proj}_{3,3}(m_2) = \mathrm{pk}(B)\ \mathit{then}$
   $K'_{AB} := \mathit{proj}_{2,3}(m_2);$
   $\mathit{send}^{11}\ \mathit{pubenc}^{12}(\mathrm{pk}(B), K'_{AB});$
   $\mathit{if}\ \mathit{isok}(\mathit{symdec}^{16}(K'_{AB}, \mathit{receive}^{17}))\ \mathit{then}$
   $\mathit{send}^{12}\ 1,$
// Participant B
   $m_1 := \mathit{pubdec}(\mathrm{sk}(B), \mathit{receive}^6);$
   $\mathit{if}\ \mathit{proj}_{2,2}(m_1) = \mathrm{pk}(A)\ \mathit{then}$
   $K_{AB} := \mathit{symkey}^7;$
   $\mathit{send}^8\ \mathit{pubenc}^9(\mathrm{pk}(A), \mathit{tuple}(\mathit{proj}_{1,2}(m_1), K_{AB}, \mathrm{pk}(B)))$
   $m3 := \mathit{pubdec}(\mathrm{sk}(B), \mathit{receive}^{13});$
   $\mathit{if}\ m_3 = K_{AB}\ \mathit{then}$
   $\mathit{send}^{14}\ \mathit{symenc}^{15}(K_{AB}, \mathit{secret}))$
```

3.2 Execution Semantics

In this section we define the structural operational semantics of the protocol. First, we define the sets and semantic functions.

In order to represent the replication dimensions of a given statement we introduce $dv \in \mathcal{P}(\mathbf{Dim})$ (note that in the protocol language each statement can have at most one coordinate in a given replication dimension). The coordinates of a statement are represented by a partial function from the replication dimensions to the coordinate values: $cv \in \mathit{CrdVect} : \mathbf{Dim} \hookrightarrow \mathbf{N}$.

The Σ is the set of all the bit strings, on which the computations are performed. Let $\tau = \Sigma \times \Sigma \to \Sigma$ be an injective function. Finally, let $T = \{\mathsf{true}, \mathsf{false}\}$ be the set of boolean values. The mathematical functions involved in computations are underlined in the semantic definitions below.

The values received from the network and generated random coins are supplied to the protocol via the following functions (both taking the label and the coordinates of the corresponding expression):

- $\mathit{inp} \in \mathit{Rand} = \mathbf{Lab} \times \mathit{CrdVect} \to \Sigma$
- $\mathit{rand} \in \mathit{In} = \mathbf{Lab} \times \mathit{CrdVect} \to \Sigma$

The current state of the computations is defined by the following sets:

- $s \in \mathit{State}$. The state is the storage for variables defined in the protocol. The variable is identified by its name and the coordinates of the statement where it has been defined. Formally, it has the type $\mathit{State} : \mathbf{Var} \times \mathit{CrdVect} \to \Sigma$;
- $t \in \mathit{Thread}$. The thread represents the single sequence of instructions (represented by S) which is yet unexecuted, along with its coordinates. Note that the thread does not hold the variable values, which are stored (globally) in $s \in \mathit{State}$. Formally, it has the type $\mathit{Thread} = (S, cv)$;
- $ts \in \mathit{Threads}$. It is a collection of the treads. Initially the protocol starts with a single thread; the additional threads are then created by the *rep* and *par* statements. Formally, it has the type $\mathit{Threads} = \mathcal{P}(\mathit{Thread})$
- $ps \in \mathit{ProtState}$. Finally, the protocol state is no more than a collection of threads and a global state. Formally, $\mathit{ProtState} = \langle ts, s \rangle$.

Given the above definitions, the semantics of the protocol are defined by the semantic function (executing the S):

$$\mathit{inp}, \mathit{rand} \vdash \mathcal{S} : \mathit{ProtState} \hookrightarrow \mathit{ProtState}$$

This function, in turn, relies on the semantics of the atomic statement, arithmetic and boolean expressions:

- Atomic statement execution: $\mathit{inp}, \mathit{rand} \vdash \mathcal{AS} : \mathbf{AStmt} \to (\mathit{CrdVect} \times \mathit{State} \to \mathit{State})$;
- Arithmetic expression evaluation: $\mathit{inp}, \mathit{rand} \vdash \mathcal{A} : \mathbf{Aexpr} \to (\mathit{CrdVect} \times \mathit{State} \to \Sigma)$;

- Boolean expression evaluation: $\mathit{inp}, \mathit{rand} \vdash \mathcal{B} : \mathbf{Bexpr} \rightarrow (\mathit{CrdVect} \times \mathit{State} \rightarrow T)$.

The semantic functions are defined as shown in the Figures 2, 3, 4, and 5. Note that the statements *send*, *begin*, and *end* do not modify the program state — *send* makes the expression value available to the adversary; *begin* and *end* (the statements are used for the examination of the integrity properties) just evaluate the given expression at the given point.

$$
\begin{array}{lcl}
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket x \rrbracket\ \mathit{cv}\ s = s\ x\ \mathit{cv} \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket n \rrbracket\ \mathit{cv}\ s = \underline{\mathit{const}}(n) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{nonce}^l \rrbracket\ \mathit{cv}\ s = \underline{\mathit{nonce}}(\mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{keypair}^l \rrbracket\ \mathit{cv}\ s = \underline{\mathit{keypair}}(\mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{pubkey}(a) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{pubkey}}(\mathcal{A}\llbracket a \rrbracket\ \mathit{cv}\ s) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{pubenc}^l(a_1, a_2) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{pubenc}}(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \\
 & & \mathcal{A}\llbracket a_2 \rrbracket\ \mathit{cv}\ s, \mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{pubdec}(a_1, a_2) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{pubdec}}(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \mathcal{A}\llbracket a_2 \rrbracket\ \mathit{cv}\ s) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{svkpair}^l \rrbracket\ \mathit{cv}\ s = \underline{\mathit{svkpair}}(\mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{verkey}(a) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{verkey}}(\mathcal{A}\llbracket a \rrbracket\ \mathit{cv}\ s) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{sig}^l(a_1, a_2) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{sig}}(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \mathcal{A}\llbracket a_2 \rrbracket\ \mathit{cv}\ s, \mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{getmsg}(a) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{getmsg}}(\mathcal{A}\llbracket a \rrbracket\ \mathit{cv}\ s) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{symkey}^l \rrbracket\ \mathit{cv}\ s = \underline{\mathit{symkey}}(\mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{symenc}^l(a_1, a_2) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{symenc}}(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \\
 & & \mathcal{A}\llbracket a_2 \rrbracket\ \mathit{cv}\ s, \mathit{rand}\ l\ \mathit{cv}) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{symdec}(a_1, a_2) \rrbracket\ \mathit{cv}\ s = \underline{\mathit{symdec}}(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \\
 & & \mathcal{A}\llbracket a_2 \rrbracket\ \mathit{cv}\ s) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{receive}^l \rrbracket\ \mathit{cv}\ s = \mathit{inp}\ l\ \mathit{cv} \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{secret}^l \rrbracket\ \mathit{cv}\ s = \mathit{rand}\ l\ \mathit{cv} \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{tuple}_n(a_1, \ldots, a_n) \rrbracket\ \mathit{cv}\ s = \tau(\mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s, \tau(\ldots, \\
 & & \tau(\mathcal{A}\llbracket a_n - 1 \rrbracket\ \mathit{cv}\ s, \mathcal{A}\llbracket a_n \rrbracket\ \mathit{cv}\ s))\ldots) \\
\mathit{inp}, \mathit{rand} & \vdash & \mathcal{A}\llbracket \mathit{proj}\, n, n'(a) \rrbracket\ \mathit{cv}\ s = \sigma_{n'}; \sigma'_0 = \mathcal{A}\llbracket a_1 \rrbracket\ \mathit{cv}\ s; \\
 & & (\sigma_m, \sigma'_m) = \tau^{-1}(\sigma'_{m-1})
\end{array}
$$

Figure 2: Semantic function $\mathcal{A}$

The *execution* of a protocol (represented by a top-level statement S), in parallel with the adversary $\mathcal{A}$ and with the secret payload M, proceeds as follows:

1. s is set to map every variable $\perp$. *rand* is initialized with the (uniformly generated) random coins used in the execution. The value $\mathit{rand}(l)$ for a

$$
\begin{aligned}
&inp, rand \vdash \mathcal{B}[\![isok(a)]\!]\; cv\; s = \begin{cases} \mathsf{true}\; \mathit{if}\; \mathcal{A}[\![a]\!]\; cv\; s \neq \perp \\ \mathsf{false}\; \mathit{if}\; \mathcal{A}[\![a]\!]\; cv\; s = \perp \end{cases} \\
&inp, rand \vdash \mathcal{B}[\![a_1 = a_2]\!]\; cv\; s = \begin{cases} \mathsf{true}\; \mathit{if}\; \mathcal{A}[\![a_1]\!]\; cv\; s = \mathcal{A}[\![a_2]\!]\; cv\; s \\ \mathsf{false}\; \mathit{if}\; \mathcal{A}[\![a_1]\!]\; cv\; s \neq \mathcal{A}[\![a_2]\!]\; cv\; s \end{cases} \\
&inp, rand \vdash \mathcal{B}[\![a_1 \neq a_2]\!]\; cv\; s = \begin{cases} \mathsf{true}\; \mathit{if}\; \mathcal{A}[\![a_1]\!]\; cv\; s \neq \mathcal{A}[\![a_2]\!]\; cv\; s \\ \mathsf{false}\; \mathit{if}\; \mathcal{A}[\![a_1]\!]\; cv\; s = \mathcal{A}[\![a_2]\!]\; cv\; s \end{cases} \\
&inp, rand \vdash \mathcal{B}[\![testsig(a_1, a_2)]\!]\; cv\; s = \underline{testsig}(\mathcal{A}[\![a_1]\!]\; cv\; s, \mathcal{A}[\![a_2]\!]\; cv\; s) \\
&inp, rand \vdash \mathcal{B}[\![b_1\, and\, b_2]\!]\; cv\; s = \begin{cases} \mathsf{true} & \mathit{if} \quad \mathcal{B}[\![b_1]\!]\; cv\; s = \mathsf{true} \wedge \\ & \qquad \mathcal{B}[\![b_2]\!]\; cv\; s = \mathsf{true} \\ \mathsf{false} & \mathit{if} \quad \mathcal{B}[\![b_1]\!]\; cv\; s = \mathsf{false} \vee \\ & \qquad \mathcal{B}[\![b_2]\!]\; cv\; s = \mathsf{false} \end{cases} \\
&inp, rand \vdash \mathcal{B}[\![b_1\, or\, b_2]\!]\; cv\; s = \begin{cases} \mathsf{true} & \mathit{if} \quad \mathcal{B}[\![b_1]\!]\; cv\; s = \mathsf{true} \vee \\ & \qquad \mathcal{B}[\![b_2]\!]\; cv\; s = \mathsf{true} \\ \mathsf{false} & \mathit{if} \quad \mathcal{B}[\![b_1]\!]\; cv\; s = \mathsf{false} \wedge \\ & \qquad \mathcal{B}[\![b_2]\!]\; cv\; s = \mathsf{false} \end{cases}
\end{aligned}
$$

Figure 3: Semantic function $\mathcal{B}$

$$
\begin{aligned}
inp, rand \quad &\vdash \quad \mathcal{AS}[\![x := a]\!]\; cv\; s = s[x \mapsto \mathcal{A}[\![a]\!]\; cv\; s] \\
inp, rand \quad &\vdash \quad \mathcal{AS}[\![send^l\, a]\!]\; cv\; s = s\; \mathit{if}\; \mathcal{A}[\![a]\!]\; cv\; s \neq \perp \\
inp, rand \quad &\vdash \quad \mathcal{AS}[\![begin^l\, a]\!]\; cv\; s = s\; \mathit{if}\; \mathcal{A}[\![a]\!]\; cv\; s \neq \perp \\
inp, rand \quad &\vdash \quad \mathcal{AS}[\![end^l\, a]\!]\; cv\; s = s\; \mathit{if}\; \mathcal{A}[\![a]\!]\; cv\; s \neq \perp
\end{aligned}
$$

Figure 4: Semantic function $\mathcal{AS}$

secret-expressions is set to M. The mapping *inp* (containing the values received by the protocol participants from the network) is set to map every value to $\perp$; the initial protocol state is set to $\langle\{(S, cv^0)\}, s\rangle$ (cv^0 maps every replication dimension to 0);

2. The adversary is invoked with the internal state it returned during the previous invocation (if this is the first invocation then the internal state is empty), the set of threads *ts* (during the first invocation it is $\{(S, cv^0)\}$), and with the value sent to the network during the last step (or $\perp$, if nothing has been sent to the network);

3. If the adversary indicates to stop, then stop the execution. Otherwise, the adversary produces a new internal state, the thread t to be executed (such that $ts = \{t\} \cup ts''$) and a new mapping inp', satisfying $inp \leq inp'$. The computational cost of outputting inp' is defined to be the number of labels l where $inp(l) \neq inp'(l)$;

$$
\begin{array}{lll}
inp, rand & \vdash & \mathcal{S}\langle(par_n(S_1, \ldots S_n), cv) \cup ts, s\rangle = \langle\bigcup_{j=1}^{n}(S_j, cv) \cup ts, s\rangle \\
inp, rand & \vdash & \mathcal{S}\langle(rep(d, S), cv) \cup ts, s\rangle = \langle\bigcup_{j\in\mathbf{N}}(S, cv[d \mapsto j]) \cup ts, s\rangle \\
inp, rand & \vdash & \mathcal{S}\langle(AS; S, cv) \cup ts, s\rangle = \langle(S, cv) \cup ts, \mathcal{AS}[\![AS]\!]\ cv\ s\rangle \\
inp, rand & \vdash & \mathcal{S}\langle(\mathit{if}\ b\ \mathit{then}\ S_1, cv) \cup ts, s\rangle = \langle(S_1, cv) \cup ts, s\rangle \\
 & & \mathit{if}\, \mathcal{B}[\![b]\!]\ cv\ s = \mathsf{true} \\
inp, rand & \vdash & \mathcal{S}\langle(\mathit{if}\ b\ \mathit{then}\ S_1, cv) \cup ts, s\rangle = \langle ts, s\rangle \\
 & & \mathit{if}\, \mathcal{B}[\![b]\!]\ cv\ s = \mathsf{false} \\
inp, rand & \vdash & \mathcal{S}\langle(\mathit{iff}\ b\ \mathit{then}\ AS_1\ \mathit{finally}\ S_1\ cv) \cup ts, s\rangle = \langle(AS_1; S_1, cv) \cup ts, s\rangle \\
 & & \mathit{if}\ \mathcal{B}[\![b]\!]\ cv\ s = \mathsf{true} \\
inp, rand & \vdash & \mathcal{S}\langle(\mathit{iff}\ b\ \mathit{then}\ AS_1\ \mathit{finally}\ S_1\ cv) \cup ts, s\rangle = \langle(S_1, cv) \cup ts, s\rangle \\
 & & \mathit{if}\ \mathcal{B}[\![b]\!]\ cv\ s = \mathsf{false} \\
inp, rand & \vdash & \mathcal{S}\langle(stop, cv) \cup ts, s\rangle = \langle ts, s\rangle
\end{array}
$$

Figure 5: Semantic function $\mathcal{S}$

4. One execution step is performed in the thread selected by the adversary. Let $\langle ts', s'\rangle = inp, rand \vdash \mathcal{S}\langle\{t\} \cup ts'', s\rangle$. If during the execution a value has been sent to the network, it is stored to be passed to the adversary in the step 2;

5. Let $s := s'$, $ts := ts'$, and $inp := inp'$. Continue from step 2.

As a result, we get a list of inputs to and outputs from the adversary. We call the probability distribution (given by the probabilistic generation of *rand* and the adversary's coin-tosses) over these lists the *view* of the adversary $\mathcal{A}$ when executed with the protocol S and the secret M. We denote this distribution by $\mathsf{view}_M^S(\mathcal{A})$.

If two protocols are indistinguishable to an adversary then they are simultaneously secure or insecure. We define the indistinguishability of protocols using the notion of adversary view. Two protocols, implemented by S_1 and S_2, are *indistinguishable*, if for all polynomial-time constructible distributions D_M of the secret M, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the distributions $\mathsf{view}_M^{S_1}(\mathcal{A})$ and $\mathsf{view}_M^{S_2}(\mathcal{A})$ are indistinguishable, if M is sampled according to D_M.

4 Dependency graphs

We represent the protocol as a *dependency graph*. The dependency graph is a directed graph, where each node corresponds to a computation, producing a *value*. The edges of the graph indicate which nodes use values produced at another nodes. The examples of the computations happening at nodes are: execution of the cryptographic algorithm, arithmetic, or boolean operation. The values produced are either bit strings or boolean values. The values produced outside of the graph (for example, random coin tosses, incoming messages, secret payloads, and constants) are brought into it via special nodes, having no incoming edges. Additionally, certain nodes explicitly make their input values available to the adversary (representing sending the values to the insecure channel).

Program dependency graphs have originated as a program analysis and optimization tool [2], systematically recording the computational relationships between different parts of a program. Since then, several flavors of dependency graphs have been proposed, some of them admitting a formal semantics [1, 3], thus being suitable as intermediate program representations in a compiler. Programs represented as dependency graphs are amenable to aggressive optimizations as all program transformations we may want to apply are incremental on dependency graphs. However, the translation from an optimized dependency graph back to a sequence of instructions executable on an actual processor may be tricky as the optimizations may have introduced patterns that are not easily serializable. The introduction of such patterns may be avoided by introducing extra dependency edges between nodes (for example, nodes corresponding to reads from and writes to the heap may depend on all previous writes to the heap) but these edges reduce the number of applicable optimizations. However we do not have this issue because we do not have to translate the optimized / simplified / analyzed protocol back to a more conventional form.

4.1 Finite Dependency Graph

A protocol, consisting of a finite number of sessions, can be represented by a *Finite Dependency Graph* (FDG). Formally, the FDG consists of:

- Finite set of nodes, each of them representing a computation. The node n is identified by:
  - $\ell(n)$ — node identity. The set of all the identities is Lab.
  - $\lambda(n)$ — an operation. An operation is either bit string-valued or boolean-valued. The operation dictates which input ports the node has.

- For each input port of each node, an edge connecting some node to this input port.

The operations, which can occur in FDG (including the input ports), are listed in the Figure 6. Each operation, having the input port cd, performs computation only if the value at this port is true; if the value is false, the $\perp$ is returned. The operation $\mathsf{Nonce}(cd, R)$ computes the bit-string representation of the random coins R. Operations $\mathsf{Keypair}(cd, R)$, $\mathsf{PubKey}(cd, K)$, $\mathsf{PubEnc}(cd, R, K, T)$, and $\mathsf{PubDec}(cd, K, T)$ abstract the asymmetric encryption scheme — they compute the public-private key pair from the random coins R, extract the public key component from the key pair K, encrypt the message T using the public key K and the random coins R, and decrypt the message T with the private key contained in the key pair T, respectively. The operation $\mathsf{PubEncZ}(cd, R, K)$ encrypts the string of zeroes using the private key from the key pair K and the the random coins R. The symmetric encryption scheme is abstracted by the operations $\mathsf{SymKey}(cd, R)$, $\mathsf{SymEnc}(cd, R, K, T)$, $\mathsf{SymDec}(cd, K, T)$, and $\mathsf{SymEncZ}(cd, R, K)$ (a string of zeros is encrypted, as in $\mathsf{PubEncZ}$). The digital signature scheme is abstracted by the operations $\mathsf{SigVer}(cd, R)$ (computation of signature-verification key pair from the random coins R), $\mathsf{VerKey}(cd, K)$ (extraction of verification key from the key pair K), $\mathsf{Signature}(cd, R, K, T)$ (signing the message T with the secret key from the key pair K and using the random coins R), $\mathsf{SignedMsg}(cd, T)$ (extraction of the message component from the signed message), and $\mathsf{TestSig}(K, M)$ (verification of the signature on the message M using the verification key K). The semantics of the operation $\mathsf{TestSigP}$ are identical to the $\mathsf{TestSig}$ (those operations are separated for the purposes of the analysis; additional details on that are supplied when the corresponding transformation is described). Tupling ($\mathsf{Tuple}^n(cd, X_1, ..., X_n)$) and projection ($\mathsf{Proj}^m_n(cd, T)$) are also supported. The operations $\mathsf{IsEq}(X_1, X_2)$ and $\mathsf{IsNeq}(X_1, X_2)$ test two values for the equality. $\mathsf{IsOK}(X)$ returns true if its argument is different from $\perp$ and false otherwise. The $\mathsf{Id}(cd, X)$ returns its argument. Error, True, and False return the constants $\perp$, true, and false, respectively.

The operation $\mathsf{Send}(cd, X)$ makes the value X (or constant $\perp$, if its control dependency is not set to true) available to the adversary. The operations $\mathsf{RS}(cd)$, $\mathsf{Const}^i(cd)$, $\mathsf{Secret}(cd)$, $\mathsf{Receive}(cd)$, and Req return the inputs to the dependency graph (computed outside of it): random coins, bit-string constant, secret payload, incoming message, and the flag, indicating the adversary's wish to evaluate certain graph output, respectively. Finally, the operations $\mathsf{Begin}(cd, X)$ and $\mathsf{End}(cd, X)$ do not perform any computation; they are used for the analysis of the integrity properties, to examine the possible order of the operations.

$$
\begin{array}{lclcl}
\lambda^f(n) & ::= & \mathsf{RS}(cd) & | & \mathsf{Nonce}(cd,R) \\
 & | & \mathsf{Const}(cd) & | & \mathsf{Keypair}(cd,R) \\
 & | & \mathsf{PubKey}(cd,K) & | & \mathsf{SigVer}(cd,R) \\
 & | & \mathsf{VerKey}(cd,K) & | & \mathsf{SymKey}(cd,R) \\
 & | & \mathsf{PubEnc}(cd,R,K,T) & | & \mathsf{SymEnc}(cd,R,K,T) \\
 & | & \mathsf{PubEncZ}(cd,R,K) & | & \mathsf{SymEncZ}(cd,R,K) \\
 & | & \mathsf{Signature}(cd,R,K,T) & | & \mathsf{SignedMsg}(cd,T) \\
 & | & \mathsf{Tuple}^n(cd,X_1,...,X_n) & | & \mathsf{Proj}^m_n(cd,T) \\
 & | & \mathsf{PubDec}(cd,K,T) & | & \mathsf{SymDec}(cd,K,T) \\
 & | & \mathsf{Send}(cd,X) & | & \mathsf{Begin}(cd,X) \\
 & | & \mathsf{End}(cd,X) & | & \mathsf{Receive}(cd) \\
 & | & \mathsf{Secret}(cd) & | & \mathsf{Merge}^n(cd,X_1,...,X_n) \\
 & | & \mathsf{Id}(cd,X) & | & \mathsf{Error} \\
 & | & \mathsf{And}^n(B_1,...,B_n) & | & \mathsf{Or}^n(B_1,...,B_n) \\
 & | & \mathsf{Req} & | & \mathsf{True} \\
 & | & \mathsf{False} & | & \mathsf{IsOK}(X) \\
 & | & \mathsf{IsEq}(X_1,X_2) & | & \mathsf{IsNeq}(X_1,X_2) \\
 & | & \mathsf{TestSig}(K,M) & | & \mathsf{TestSigP}(K,M)
\end{array}
$$

Figure 6: FDG operations

To illustrate the FDG concept, let us represent the Needham-Schroeder-Lowe key-exchange protocol as an FDG. The protocol from the previous section (1), where only these five messages are sent (i.e. there is only one session for A in the role of the initiator and one session for B in the role of the responder) is depicted as a FDG in Figure 7. In this figure, at the left of each node n is written the operation $\lambda(n)$ that it performs; to the right is the unique identifier $\ell(n)$ of that node.

The specification of the protocol starts with generating the public encryption and secret decryption key for both A and B. The key pairs are generated at the nodes 3 (for A) and 15 (for B); the public key is extracted from the pairs (nodes 12 and 24).

To produce the first message from A, a nonce has to be generated, this occurs at node 27. We then form a pair of the generated nonce and the identity of A (represented by the public key of A). This pair is constructed at node 69. At node 72 it is then encrypted with the public key of B. The resulting cipher text is sent to the network at the node 78. Note that we do not specify the intended recipient here, as the network is under the full control of the adversary.

The node for receiving the first message at B has the identity 165 (in the semantics of dependency graphs, the adversary controls the value produced

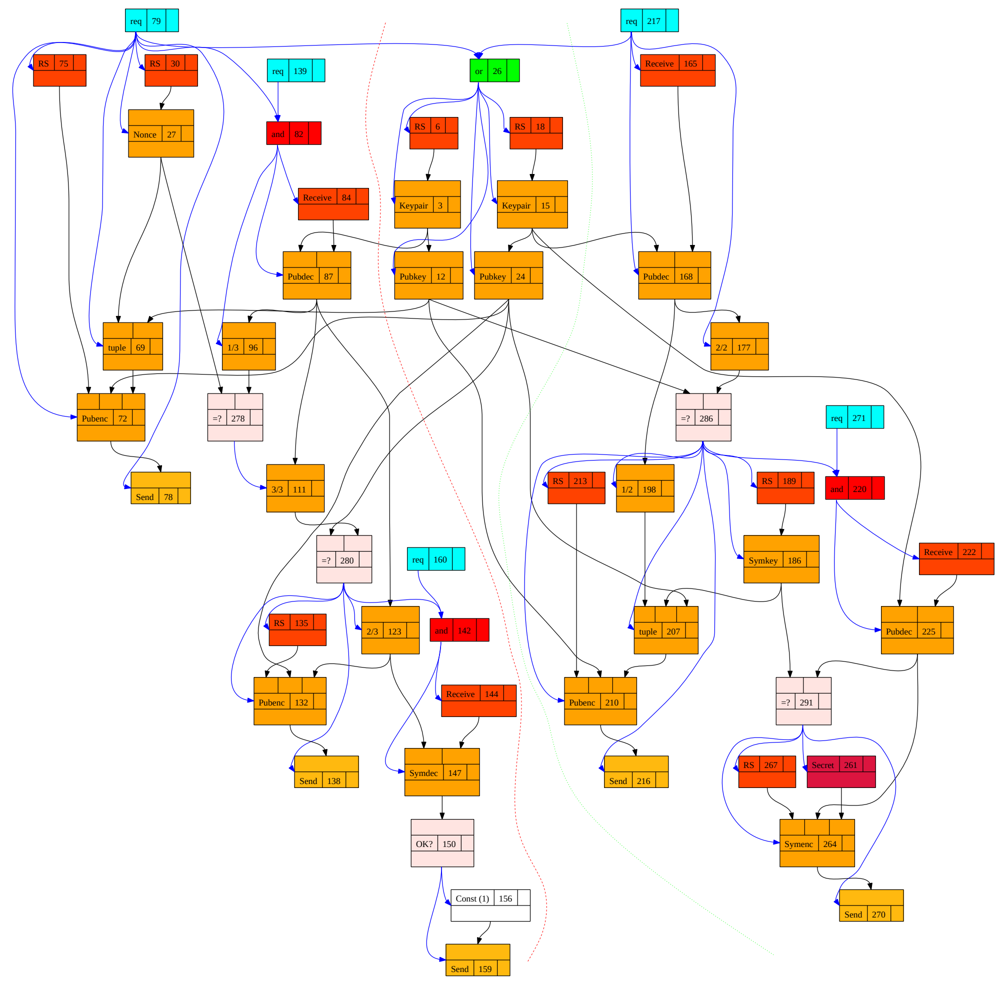
Figure 7: Single session of Needham-Schroeder-Lowe key-exchange protocol

by it). This message is then decrypted using the secret decryption key (represented by the key pair) of B. The decryption (or possibly any other operation) may fail; if any node receives failure as one of its inputs, the result is failure, too (except at certain special cases). Next, B checks that the decrypted message claims to originate from A (note that this check is not explicit in (1)) — it takes the second component of a 2-tuple and checks at node 286 whether it is equal to the public key of A. Node 286 produces a boolean value as a result (all other nodes we have mentioned so far produce bit-strings). If any of the inputs to node 286 is a failure, the result is false. In Figure 7 the black edges carry bit-strings while the blue edges carry booleans. Each node producing bit-strings also has a special input — the boolean-valued *control dependency.* A node is evaluated only if its control dependency is true; the result of any node with false control dependency is a failure.

We see that the rest of the construction of the second message of the protocol is performed only if the test at node 286 returns true. We extract the first part of the 2-tuple at node 198 (this should equal N_A), generate the shared key K_{AB} at node 186, form the triple (N_A, K_{AB}, B) at node 207 (again, the identity of B is represented by his public key), encrypt it and send it away.

Node 84 is supposed to receive the second message sent by B (if the adversary forwards it directly to A). We see that this message is decrypted using the public key of A, the first component of the plain text (which is supposed to be a triple) is compared against the nonce N_A and the third component is compared against the identity of B. If both these checks succeed then the third message is constructed by encrypting the second component of the received plain text triple. The message is then sent away.

Note that although the control dependency of the nodes constructing and sending the third message (nodes 123, 132, 135 and 138) comes just from node 280, their execution also depends on the result of the comparison at node 278. Indeed, node 278 produces the control information for node 111, and if the result of node 278 is false, then node 111 produces a failure and node 280 returns false, too (note also that comparing failure to failure results in false, too).

The rest of the graph depicted in Figure 7 (apart from the nodes with the operation Req) represents the reception of the third and fourth message and the construction of the fourth and fifth message. The secret payload is represented by a node with the operation Secret. After receiving the fourth message at node 144, the party A attempts to decrypt it and if the decryption succeeds emits the acknowledgement message (which is a constant).

When are the depicted operations actually performed, i.e. how are the parties A and B scheduled? We take a common approach to scheduling by giving the adversary the control over it [7, 4]. The adversary performs the scheduling by *requesting* the parties to deliver a certain message. For receiving those requests, the protocol contains the nodes with the operations Req. The value of these nodes is false initially, but the adversary may set them to true in the course of the computation. Setting some of the Req-nodes to true may result in the control dependencies of some Send-nodes becoming true, too, and the values reaching those nodes are then sent to the adversary. The adversary sees which Send-nodes are the sources of messages. In our examples, there is a one-to-one correspondence between Send-nodes and Req-nodes. This is indeed a natural way for representing protocols, although not strictly necessary from the semantics point of view.

The graph in Figure 7 corresponds to just a single session by A as the initiator and a single session by B as the responder. To model more sessions we simply have to add more nodes to the graph. For example, to have A act as the initiator in three sessions of the protocol, all nodes to the left of the dashed red line in Figure 7 have to be included three times. Similarly, to have B act as the responder in two sessions of the protocol, all nodes to the right of the dashed green line have to be included twice. The edges between those nodes connect nodes belonging to the same copy. Each edge from the central part (where the public key pairs are generated) to the left will become three edges, going to each of the copied nodes, while each edge from the central part to the right will become two edges. In Figure 7 there is just a single edge from the left part to the center (from node 79 to node 26) and a single edge from the right part to the center (from node 217 to node 26). In the graph with multiple sessions, these edges are replaced by multiple edges as well.

4.2 Infinite Dependency Graph

As dependency graphs defined above contain a node for each operation that a party may want to perform, the graphs representing unbounded number of rounds of a protocol may be infinite.

The *Infinite Dependency Graph* (IDG) is still defined as a collection of nodes and edges. The important difference from the FDG is that each node may also have one or more coordinates in so-called replication dimensions (the coordinate corresponds to the index of a session a party is running).

Being infinite, the graph stays regular enough to be represented in the finite form. We call this form the *Infinite Dependency Graph Representation* (IDGR).

Certain additional operations, not occurring in FDG, are possible — the full list of IDGR operations is supplied in Figure 8. The Longor is the Or with an infinite number of inputs. IfDef and CfDef-operations are multiplexors, having potentially infinite number of selector-value pairs, and returning the value, the selector of which is set to true. The difference between them is that the CfDef operates on boolean values, while IfDef — on bit strings.

The above operations, in addition to the input ports, have another kind of important attribute — the dimensions among which the coordinates of the inputs may range (it will be explained in more details on the example below). The DimEq and DimNeq represent the set of boolean nodes, True or False depending on whether the values of the given coordinates of the node are equal or not. Finally, the $\mathsf{CTakeDimEq}$ (operating on boolean values) and $\mathsf{DTakeDimEq}$ (operating on bit strings) correspond to the set of And[1] (in case of $\mathsf{CTakeDimEq}$) or Id (in case of $\mathsf{DTakeDimEq}$) operations, having its inputs with two certain coordinates being equal.

$$\begin{array}{rcll}
\lambda^i(n) & ::= & \lambda^f(n) & \mid \ \mathsf{Longor}(B) \\
& \mid & \mathsf{IfDef}^n(cd, B_1, X_1, ..., B_n, X_n) & \mid \ \mathsf{CfDef}^n(cd, B_1, B_1', ..., B_n, B_n') \\
& \mid & \mathsf{DimEq} & \mid \ \mathsf{DimNeq} \\
& \mid & \mathsf{DTakeDimEq}(X) & \mid \ \mathsf{CTakeDimEq}(X)
\end{array}$$

Figure 8: FDG operations

As an example, consider the implementation of the Needham-Schroeder-Lowe protocol (1), where

- A is able to run an unbounded number of sessions as the initiator with the principal B (i.e. A makes use of B's public key);
- B is able to run an unbounded number of sessions as the responder with the principal A;
- A is able to run an unbounded number of sessions as the initiator with unbounded number of other parties whose public key is given to it (under the control of the adversary).

This configuration is interesting because the Lowe's attack is applicable to the original Needham-Schroeder protocol [6] in this setting. Our representation of the (infinite) dependency graph representing such execution of the NSL-protocol is given in Figure 9.

We see that the main addition to the plain dependency graphs are the *replication dimensions* depicted at the right of the nodes. A node with $k \geq 1$ dimensions represents a countable number of nodes in the actual dependency graph; these nodes are in one-to-one correspondence with the elements of $\mathbf{N}^k$. In Figure 9, the dimension AtoB corresponds to the sessions where A talks to B, and BtoA to the sessions where B talks to A. The dimension AtoOthers accounts for the other parties that A may talk to, the public key of the other party is obtained at node 163. The dimension AtoOne corresponds to the

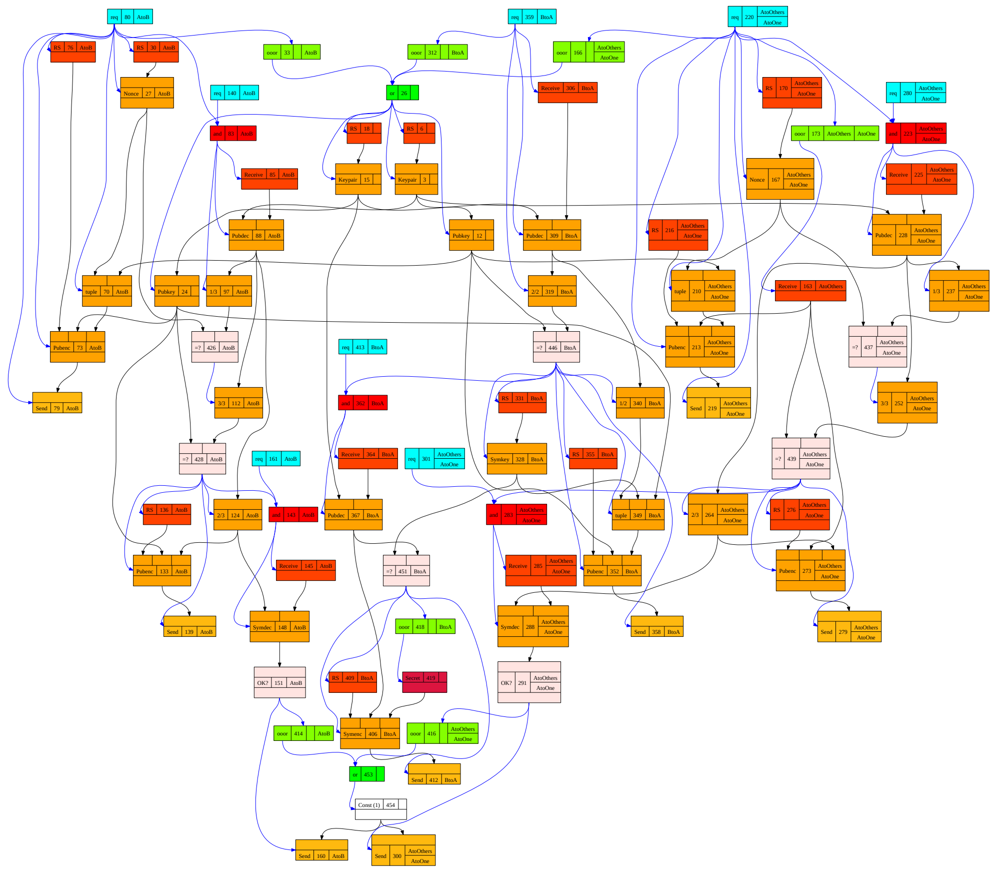
Figure 9: Unbounded number of sessions for the Needham-Schroeder-Lowe protocol

sessions where A talks to one of those parties. An edge between two nodes in the representation corresponds to a set of edges in the actual dependency graph between the nodes with the same coordinates. If the source of the edge has less dimensions at the representation than its target, then each of the corresponding source nodes in the actual dependency graph is connected to an infinite number of target nodes.

Figure 9 contains a kind of nodes that were not present in Figure 7. These are the nodes with the operation Longor (disjunctions with infinite number of inputs). The replication dimensions of these nodes are depicted at the second slot from the right (the dimension of node 173 is AtoOthers and the set of dimensions of other Longor-nodes is empty) while the rightmost slot lists the dimensions that these nodes contract. In the actual dependency graph, the Longor-nodes correspond to Or-nodes whose set of input edges is infinite. In Sec. 6 we will describe another nodes (IfDef, CfDef) that may have an infinite number of inputs in the actual dependency graph.

Formally, a node n in the IDGR has the following attributes:

- Its identity $\ell(n)$;
- The operation it performs $\lambda^i(n)$. As it is the case with FDG, the operation determines the input ports of the node;
- Its replication dimensions — a mapping $r(n) : \mathbf{D} \rightarrow \mathbf{N}$;
- For special nodes
  - Longor-nodes: the dimensions it contracts — a mapping $c(n) : \mathbf{D} \rightarrow \mathbf{N}$;
  - IfDef^n- and CfDef^n-nodes: for each input port pair (B_i, X_i) the dimensions it contracts — a mapping $c_i(n) : \mathbf{D} \rightarrow \mathbf{N}$;
  - DimEq- and DimNeq-nodes: the dimension D_c and a two indices c_1 and c_2 of the coordinates compared;
  - DTakeDimEq(X)- and CTakeDimEq(X)-nodes the dimension D_c and an index c_1 of the coordinate compared with the coordinate $r(D_c)$.

Each edge e of the IDGR has the following attributes:

- Its source and target nodes $s(e)$ and $t(e)$;
- For each $D \in \mathbf{D}$: a mapping $m_D(e)$ from $\{1, \ldots, r(s(e))(D)\}$ to $\{1, \ldots, \mathit{inpdims}_{t(e)(D)}\}$ where $\mathit{inpdims}_n$ equals
  - $r(n)$ for nodes with operations from λ^f, DimEq- and DimNeq-nodes;
  - $r(n) + c(n)$ (pointwise addition) for Longor-nodes;

  - $r(n) + c_i(n)$ (pointwise addition) for i-th input port of IfDef- and CfDef-nodes;
  - $r(n) + (D_c, 1)$ for DTakeDimEq(X), CTakeDimEq(X)-nodes.

IDGR is translated to IDG in the following way:

- For each node n in the representation there are nodes $n[i_1^{D_1},\dots,i_{r1}^{D_1},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k}]$ in the dependency graph, where $\{D_1,\dots,D_k\} = \mathbf{D}$, $r_i = r(n)(D_i)$, and $i_1^{D_1},\dots,i_{r_1}^{D_1},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k} \in \mathbf{N}$. All these nodes have the same operation as n, unless the operation of n is:
  - Longor, in which case the operation of the corresponding nodes in the IDG is Or;
  - DimEq, in which case the operation is: True (if $i_{c_1}^{D_c} = i_{c_2}^{D_c}$) or False (if $i_{c_1}^{D_c} \neq i_{c_2}^{D_c}$);
  - DimNeq, in which case the operation is False (if $i_{c_1}^{D_c} = i_{c_2}^{D_c}$) or True (if $i_{c_1}^{D_c} \neq i_{c_2}^{D_c}$);
  - DTakeDimEq, in which case the operation is Id;
  - CTakeDimEq, in which case the operation is And[1];
- For n' with $\lambda^i(n') \notin \{\mathsf{CTakeDimEq}, \mathsf{DTakeDimEq}\}$:
  - For each edge e from n to the port p of n' and each coordinate vector $(i_1^{D_1},\dots,i_{r_1}^{D_1},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k})$ there is an edge from $n[i_1^{D_1},\dots,i_{r_1}^{D_1},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k}]$ to the port p of the nodes $n'[j_1^{D_1},\dots,j_{r_1}^{D_1},\dots,j_1^{D_k},\dots,j_{r_k}^{D_k}]$, where
    - $j_{m_D(e)(l)}^D = i_l^D$ for all $D \in \mathbf{D}$ and $l \in \{1,\dots,r(n)(D)\}$;
    - the rest of the coordinates of n' vary freely over $\mathbf{N}$.
- For n' with $\lambda^i(n') \in \{\mathsf{CTakeDimEq}, \mathsf{DTakeDimEq}\}$:
  - For each edge e from n to the port p of n' and each coordinate vector $(i_1^{D_1},\dots,i_{r_1}^{D_1},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k})$ there is an edge from $n[i_1^{D_1},\dots,i_{r_1}^{D_1},\dots,i_1^{D_c},\dots,i_{c_1}^{D_c},\dots,i_{r_{D_c}}^{D_c} = i_{c_1}^{D_c},\dots,i_1^{D_k},\dots,i_{r_k}^{D_k}]$ to the port p of the nodes $n'[j_1^{D_1},\dots,j_{r_1}^{D_1},\dots,j_1^{D_k},\dots,j_{r_k}^{D_k}]$, where
    - $j_{m_D(e)(l)}^D = i_l^D$ for all $D \in \mathbf{D}$ and $l \in \{1,\dots,r(n)(D)\}$;
    - the rest of the coordinates of n' vary freely over $\mathbf{N}$.

4.3 Execution Semantics

To speak about the correctness of transformations, we have to specify what it means to execute a dependency graph. Informally, the execution starts with the tossing of coins for all RS-nodes. Also, the secret payload(s) is/are fixed. The current values for all nodes are set to $\perp$, denoting failure (for nodes computing bit-strings) or false (for nodes computing booleans). The following steps are then repeatedly executed, until the adversary decides to break the cycle:

1. The adversary will set the value of some Req-nodes to true and/or set (but not change) the computed values for some Receive-nodes.
2. The operations in the nodes are performed as long as possible. Technically, a certain least fixed point is computed. The fixed points exist because the semantics of all operations is monotone.
3. The values of all Send-nodes (or: all Send-nodes that changed from $\perp$ to some bit-string) are given to the adversary.

Formally, let Lab be the set of all nodes of the dependency graph. Let Lab_* be the subset of Lab containing all nodes with operations RS or Secret. Let $\mathit{Lab}_\leftarrow \subseteq \mathit{Lab}$ be the set of all nodes with operations Receive or Req. Let $\mathit{Lab}_\rightarrow$ be the set of all nodes with operations Send. Let $\mathit{Lab}_\bullet = \mathit{Lab}\backslash(\mathit{Lab}_\leftarrow \cup \mathit{Lab}_*)$, i.e. $\mathit{Lab}_\bullet$ contains all nodes whose value is computed in the "usual" way, not making any use of some outside information.

Let $\mathbb{B} = \{\mathsf{false}, \mathsf{true}\}$, ordered with $\mathsf{false} \leq \mathsf{true}$. Let $\Sigma = \{0,1\}^*$ and $\Sigma_\perp = \Sigma \,\dot{\cup}\, \{\perp\}$ ordered with $\perp \leq x$ for all $x \in \Sigma$. Let $\mathbf{Val} = \mathbb{B} \,\dot{\cup}\, \Sigma_\perp$. A *configuration* of a dependency graph is a triple $\langle \rho, \phi, \psi \rangle \in \mathbf{Conf}$, where

- $\rho : \mathit{Lab} \rightarrow \mathbf{Val}$ gives the current value of each node;
- $\phi : \mathit{Lab}_\leftarrow \rightarrow \mathbf{Val}$ gives the values that have been set by the adversary;
- $\psi : \mathit{Lab}_* \rightarrow \mathbf{Val}$ gives the values that are set during the initialization.

In an initial configuration, ρ and ϕ map all nodes to $\perp$ or false (depending on the types of operations in nodes).

All operations have associated semantic functions. For example,

$$[\![\mathsf{PubEnc}]\!](cd, R, K, T) = \begin{cases} \perp, & \text{if } cd = \mathsf{false} \\ \perp, & \text{if some of } R, K, T \text{ equals } \perp \\ \mathcal{E}^R(K, T), & \text{if } cd = \mathsf{true}, R, K, T \in \Sigma, \end{cases}$$

where $\mathcal{E}$ is the actual algorithm implementing the public-key encryption, using R as the randomness, K as the key and T as the plain text. In particular, the semantics of all operations of nodes in $Lab_{\bullet}$ computing with bit-strings is strict, i.e. it returns $\perp$ / false unless all of its inputs are different from $\perp$ / false. In particular, $[\![\mathsf{IsEq}]\!](\perp, \perp) = \mathsf{false}$.

The semantics of most of the operations in nodes in $Lab_{\bullet}$ should be obvious — there has to be an algorithm with the correct number of inputs; this algorithm is invoked whenever all incoming edges of the node carry non-bottom values. We mention only that we assume that all message constructors (key generation, public-key extraction, tupling, encryption, nonce generation) tag the resulting bit-string with its type, such that two bit-strings produced by different types of constructors are always different.

Each node n is associated with a *step function* $f^n : \mathbf{Conf} \rightarrow \mathbf{Val}$. For $n \in Lab_{\bullet}$, the evaluation of $f^n(\langle \rho, \phi, \psi \rangle)$ extracts the values $\rho(n_i)$ for the source nodes n_i of the edges ending at n, and applies $[\![\lambda(n)]\!]$ to them. As the functions $[\![op]\!]$ are all monotone, the step functions f^n are also monotone.

If $\lambda(n) = \mathsf{Receive}$ and u is the source node for the control dependency of n, then $f^n(\langle \rho, \psi, \phi \rangle)$ is $\phi(n)$ if $\rho(u) = \mathsf{true}$, and $\perp$ otherwise. If $\lambda(n) = \mathsf{Req}$ then $f^n((\langle \rho, \psi, \phi \rangle) = \phi(n)$. If $v \in Lab_{*}$ and u is the source node for n's control dependency then $f^n(\langle \rho, \psi, \phi \rangle)$ equals either $\psi(n)$ or $\perp$, depending on whether $\rho(u)$ is true or false.

The parallel application of all the step functions for nodes gives us a step function of the whole configuration:

$$f(\langle \rho, \phi, \psi \rangle) = \langle \rho[v \mapsto f^v(\langle \rho, \phi, \psi \rangle)], \phi, \psi \rangle \ .$$

The function f is monotone and continuous.

The *execution* of a dependency graph, in parallel with the adversary $\mathcal{A}$ and with the secret payload M, proceeds as follows:

1. ρ is set to map every dependency to $\perp$ / false. ψ is initialized with the (uniformly generated) random coins used in the execution. For each v with $\lambda(v) = \mathsf{RS}$, the value $\psi(v)$ is a sufficiently long random bit-string that is independent from every other value. The value $\psi(v)$ for a Secret-node is set to M. The mapping ϕ (containing information on which of the protocol outputs are to be evaluated and the values to be fed to the graph from the network) is set to map every Req operation to false and every $\mathsf{Receive}$ operation to $\perp$.

2. The adversary is invoked with the internal state it returned during the previous invocation (if this is the first invocation then the internal state is empty) and with the mapping $\rho|_{Lab_{\rightarrow}}$.

3. If the adversary indicates to stop, then stop the execution. Otherwise, the adversary produces a new internal state and a new mapping ϕ' :

$Lab_{\leftarrow} \to \Sigma_{\perp}$ satisfying $\phi \leq \phi'$. The computational cost of outputting ϕ' is defined to be the number of labels l where $\phi(l) \neq \phi'(l)$.

4. The graph is evaluated — let $\langle \rho', \phi', \psi \rangle$ be the least fixed point of f that is greater or equal to $\langle \rho, \phi', \psi \rangle$. The existence of such fixed point follows from the properties of f.

5. Let $\rho := \rho'$, $\phi := \phi'$. Continue from step 2.

As a result, we get a list of inputs to and outputs from the adversary. We call the probability distribution (given by the probabilistic generation of ψ and the adversary's coin-tosses) over these lists the *view* of the adversary $\mathcal{A}$ when executed with the given dependency graph G and the secret M. We denote this distribution by $\mathsf{view}_M^G(\mathcal{A})$.

4.4 Indistinguishability of Protocols

If two protocols (both implemented as dependency graphs) are indistinguishable to an adversary then they are simultaneously secure or insecure. Having defined the notion of adversary's view, it is easy to define the indistinguishability of protocols. Two protocols, implemented by G_1 and G_2 are *indistinguishable* if for all polynomial-time constructible distributions D_M of the secret M, and all probabilistic polynomial-time adversaries $\mathcal{A}$, the distributions $\mathsf{view}_M^{G_1}(\mathcal{A})$ and $\mathsf{view}_M^{G_2}(\mathcal{A})$ are indistinguishable, if M is sampled according to D_M.

5 Translation from the Protocol Language to DG

In order to analyze the protocol, we need to represent it as a DG first. The common way of describing a protocol is a procedural language. We use the language, defined in sec. 3.1. In this section we describe how to compose a DG, corresponding to a given protocol, expressed in the procedural language.

In the process of translating a protocol, specified in the procedural language, to the DGR, two functions are used:

- Expression translation (function *EtoDG*);
- Atomic statement translation (function *AStoDG*);
- Compositional statement translation (function *StoDG*).

Translating a protocol (represented by a top-level statement) is performed by application of *StoDG* to it. Additionally, *StoDG* takes the DGR, generated so far, as an argument (for the initial call, the empty DGR). If the statement processed by the *StoDG* consists of several statements, it recursively calls itself for translating each of them (passing the statement to translate and the DGR, generated so far, as an argument). During the translation of a statement, the functions *AStoDG* and *EtoDG* are called (the DGR, generated so far, is passed as an argument).

The rest of this section contains the detailed specification of the translation functions, proofs for their correctness (with respect to the equivalence of the resulting DG semantics and the semantics of the statement or the expression being translated), and the formal algorithm for translating the protocol by applying the mentioned functions.

5.1 *EtoDG* — Translation from Aexpr / Bexpr to DGR

Let $\mathbf{Expr} = \mathbf{Aexpr} \mathbin{\dot\cup} \mathbf{Bexpr}$. Formally, the function *EtoDG* has type:

$$\begin{aligned} \mathit{EtoDG} \quad : \quad & \mathbf{Expr} \\ \rightarrow \quad & (\mathcal{P}(\mathbf{Dim}) \times \mathbf{GR} \times (\mathbf{Var} \rightarrow (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})) \times \mathbf{Lab} \\ \rightarrow \quad & (\mathbf{GR} \times \mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})) \end{aligned}$$

The arguments to the *EtoDG* function are:

- $e \in \mathbf{Expr}$: expression to translate;
- $dv \in \mathcal{P}(\mathbf{Dim})$: function specifying the replication dimensions the expression should have a coordinate in;
- $gr \in \mathbf{GR}$: DGR, containing the nodes for calculating the variables which may be used in e and the continuation node;

- $vartonodes \in \mathbf{Var} \rightarrow (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})$: Function returning for each of the free variables in e the labels of three nodes — so called Or-node, And-node, and *val*-node. The requirements put on those nodes are discussed below;

- $cn \in \mathbf{Lab}$: the continuation node. The dependency, the value of which should be equal to true, in order for the expression to be evaluated.

The DGR passed to *EtoDG* as an argument should contain all the operations for each variable in $FV(e)$. For each variable there are three sets of nodes, the labels of which are returned by *vartonodes*. Suppose that for some $v \in FV(e)$ the $vartonodes\, v = ea\ eo\ en$. These nodes are in a certain relation between themselves and the semantics of the protocol corresponding to the gr. The relation is following. Given that

- A protocol, corresponding to gr, is currently in configuration $\langle ts, s\rangle$,

- on the dependency graph, corresponding to gr, $\rho[eo.cv]$ is set to true,

- the dependency graph is evaluated,

the following holds:

- The $\rho[ea.cv] =$ true,

- The $\rho[en.cv] = s\ v\ cv$.

Intuitively, the graph, specified by gr, corresponds to the part of the protocol, preceding the evaluation of e. Additional condition we put on the gr is that the $\rho[cn]$ (with the corresponding coordinate vector cv', which is (by the number of coordinates in each dimension) less than or equal to cv) should be equal to true. This condition (established and maintained in *StoDG* and used in *EtoDG*) ensures that all the invariants set by the preceding *if* and *iff* statements, are checked prior to the expression evaluation.

The general idea behind the *EtoDG* is the following. Given the DGR gr, and node cn, which (with the coordinate vector cv', corresponding to cv) is guaranteed to be true when the protocol is executed to the point when the expression e (at the statement with coordinate vector cv) is evaluated (let this state be $\langle ts, s\rangle$, given the adversary input inp, the random coins $rand$), and the triple of nodes for each of the variables free in e, we construct a new graph gr', which is equal to gr, but has the triple of nodes with labels vo, va, vn such that if $\rho[vo.cv]$ is set to true, and the graph gr' is evaluated, the $\rho[va.cv] =$ true, and the $\rho[vn.cv] = inp, rand \vdash \mathcal{A}[\![e]\!]\ cv\ s$, given that $\phi = inp$ and $\psi = rand$. The definition of *EtoDG* (as well as the structure of expressions it handles) is strictly compositional — it calls itself for generating the dependency graphs corresponding to the sub-expressions the expression e is composed of, before constructing the graph for e.

Evaluation of $EtoDG[e]\ dv\ gr\ vartonodes\ cn$ results in the gr' and the labels o, a, v, and proceeds in the following way, depending what the e is:

1. e is x. As $x \in FV(e)$, $vartonodes\,x$ is defined. Let $(x^a, x^o, x^v) = vartonodes\,x$. As it will be shown during the definition of *StoDG*, the nodes x^a, x^o, and x^v have the same number of coordinates. Also, it will be shown during the definition of *StoDG*, that during the protocol translation, the replication dimensions can only be extended, so the number of coordinates in each dimension for every operation present in the gr will be smaller than or equal to dv. The result of the translation depends on whether the x^o has equal amount or less coordinates, than specified by dv:

   (a) If the node x^o has same replication dimensions as given by dv, then the gr' is equal to gr, with the following changes:
   - The node x^o (having n inputs; let the i-th input be x^{os^i}, with the coordinate mapping λ^i) is removed;
   - The node with label o, the operation Or, and replication dimensions dv is added. So far the node has no inputs;
   - The node with label a, the operation And, and replication dimensions dv is added. The node inputs are:
     - cn, with identity coordinate mapping;
     - o, with identity coordinate mapping;
     - x^a, with identity coordinate mapping;
   - The node v, the operation Id, and replication dimensions dv. The node control input is a, with identity coordinate mapping. The data input is x^v, with identity coordinate mapping;
   - The node x^o, operation Or, replication dimensions dv, and $n+1$ inputs, is added. For $1 \le i \le n$ the i-th input is x^{os^i}, with the coordinate mapping λ^i; the $n+1$-th input is o, with identity coordinate mapping.

   (b) If the number of x^o dimensions, given by dv', is strictly less than the dv, then the gr' is equal to gr, with the following changes:
   - The node x^o (having n inputs; let the i-th input be x^{os^i}, with the coordinate mapping λ^i) is removed;
   - The node o, the operation Or, and replication dimensions dv, is added. So far the node has no inputs;
   - The node a, the operation And, and replication dimensions dv, is added. The node inputs are:

    - cn, with identity coordinate mapping;
    - o, with identity coordinate mapping;
    - x^a, with identity coordinate mapping;
  - The node lo, the operation Longor, and replication dimensions dv', is added. The input is o, with identity coordinate mapping. All the dv coordinates, not existing dv', are contracted;
  - The node v, the operation Id, and replication dimensions dv, is added. The node control input is a, with identity coordinate mapping. The data input is x^v, with identity coordinate mapping;
  - The node x^o, operation Or, replication dimensions dv', is added. The node has $n+1$ inputs: for $1 \leq i \leq n$ the i-th input is x^{os^i}, with the coordinate mapping λ^i; the $n+1$-th input is lo, with identity coordinate mapping.

2. e is an arithmetic expression taking zero or more arguments, and not using the random coins. Out of the expressions we supplied in the protocol language, the following operations fall into this group: n, $secret^l$, $receive^l$ (0 arguments), $pubkey(a)$, $verkey(a)$, $getmsg(a)$, $proj_{n,n'}(a)$ (1 argument), $pubdec(a_1, a_2)$, $symdec(a_1, a_2)$ (2 arguments), and $tuple_m(a_1, \ldots, a_m)$ (m arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i. First, the expression arguments are processed in the following way:

$$\begin{array}{rcl} gr^1, o^1, a^1, v^1 & = & EtoDG[a_1]\ dv\ gr\ vartonodes\ cn \\ & \cdots & \\ gr^n, o^n, a^n, v^n & = & EtoDG[a_n]\ dv\ gr^{n-1}\ vartonodes\ cn \end{array}$$

   The result of the translation is gr^n, with the following changes:

   - The nodes o^i for $1 \leq i \leq n$ (having no inputs, as they has just been created during the $EtoDG$ invocation) are removed;
   - The node o, the operation Or, and replication dimensions dv, is added. So far the node has no inputs;
   - The node a, the operation And, and replication dimensions dv, is added. The node has $n+2$ inputs:
     - cn, with identity coordinate mapping;
     - o, with identity coordinate mapping;
     - for $1 \leq i \leq n$, a^i, with identity coordinate mapping;

- The node v, the operation corresponding to the expression being processed (e.g. Const for n, Tuple for $tuple_n$, and so on), and replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The node has n data inputs: for $1 \leq i \leq n$, v^i, with identity coordinate mapping; Note that the label v is chosen to be equal to l (the label of the expression);
- The nodes o^i for $1 \leq i \leq n$, with the operation Or, replication dimensions dv, and a single input o, with identity dimension mapping, are added.

3. e is an arithmetic expression taking zero or more arguments, and using the random coins. Out of the expressions we supplied in the protocol language, the following fall into this group: $nonce^l$, $keypair^l$, $symkey^l$, $svkpair^l$ (0 arguments and random coins), $pubenc^l(a_1, a_2)$, $symenc^l(a_1, a_2)$, $sig^l(a_1, a_2)$ (2 arguments and random coins).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i. First, the expression arguments are processed in the following way:

$$\begin{aligned} gr^1, o^1, a^1, v^1 &= EtoDG[a_1]\ dv\ gr\ vartonodes\ cn \\ &\dots \\ gr^n, o^n, a^n, v^n &= EtoDG[a_n]\ dv\ gr^{n-1}\ vartonodes\ cn \end{aligned}$$

   The result of the translation is gr^n, with the following changes:

   - The nodes o^i for $1 \leq i \leq n$ (having no inputs, as they has just been created during the $EtoDG$ invocation) are removed;
   - The node o, the operation Or, and replication dimensions dv, is added. So far the node has no inputs;
   - The node r^o, the operation Or, and replication dimensions dv, is added. The node input is o, with identity coordinate mapping;
   - The node r^a, the operation And, and replication dimensions dv is added. The node inputs are:
     - cn, with identity coordinate mapping;
     - r^o, with identity coordinate mapping;
   - The node r^v, the operation is RS. The replication dimensions are dv. The node control input is r^a, with identity coordinate mapping. Note that the label r^v is chosen to be equal to l (the label of the expression);

- The node a, the operation And, and replication dimensions dv, is added. The node has $n+3$ inputs:
  - cn, with identity coordinate mapping;
  - o, with identity coordinate mapping;
  - for $1 \leq i \leq n$, a^i, with identity coordinate mapping;
  - l^{ra}, with identity coordinate mapping;
- The node v, the operation corresponding to the expression being processed (e.g. Nonce for *nonce*, PubEnc for *pubenc*, and so on), and replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The node has $n+1$ data inputs: the r^v (random coins input), with identity coordinate mapping, and, for $1 \leq i \leq n$, v^i, with identity coordinate mapping;
- The nodes o^i for $1 \leq i \leq n$, with the operation Or, replication dimensions dv, and a single input o, with identity dimension mapping, are added.

4. e is a boolean expression taking arithmetic arguments. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $isok(a)$ (1 argument), $a_1 = a_2$, $a_1 \neq a_2$, $testsig(a_1, a_2)$ (2 arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i. First, the expression arguments are processed in the following way:

$$\begin{aligned} gr^1, o^1, a^1, v^1 &= EtoDG[a_1]\ dv\ gr\ vartonodes\ cn \\ &\dots \\ gr^n, o^n, a^n, v^n &= EtoDG[a_n]\ dv\ gr^{n-1}\ vartonodes\ cn \end{aligned}$$

   The result of the translation is gr^n, with the following changes:

   - The nodes o^i for $1 \leq i \leq n$ (having no inputs, as they has just been created during the *EtoDG* invocation) are removed;
   - The node o, the operation Or, and replication dimensions dv, is added. So far the node has no inputs;
   - The node a, the operation And, and replication dimensions dv, is added. The node has $n+2$ inputs:
     - cn, with identity coordinate mapping;
     - o, with identity coordinate mapping;
     - for $1 \leq i \leq n$, a^i, with identity coordinate mapping;

- The node v', the operation corresponding to the expression being processed (e.g. IsOK for *isok*, etc.), and replication dimensions dv, is added. The node has n data inputs: for $1 \leq i \leq n$, v^i, with identity coordinate mapping;
- The node v, And operation, replication dimensions dv, and two inputs: a and v', both with identity coordinate mappings, is added;
- The nodes o^i for $1 \leq i \leq n$, with the operation Or, replication dimensions dv, and a single input o, with identity dimension mapping, are added.

5. e is a boolean expression taking boolean arguments. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $and(b_1, \ldots, b_m)$, $or(b_1, \ldots, b_m)$ (m arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i. First, the expression arguments are processed in the following way:

$$\begin{aligned} gr^1, o^1, a^1, v^1 &= EtoDG[b_1]\ dv\ gr\ vartonodes\ cn \\ &\ldots \\ gr^n, o^n, a^n, v^n &= EtoDG[b_n]\ dv\ gr^{n-1}\ vartonodes\ cn \end{aligned}$$

   The result of the translation is gr^n, with the following changes:

   - The nodes o^i for $1 \leq i \leq n$ (having no inputs, as they has just been created during the *EtoDG* invocation) are removed;
   - The node o, the operation Or, and replication dimensions dv, is added. So far the node has no inputs;
   - The node a, the operation And, and replication dimensions dv, is added. The node has $n + 2$ inputs:
     - cn, with identity coordinate mapping;
     - o, with identity coordinate mapping;
     - for $1 \leq i \leq n$, a^i, with identity coordinate mapping;
   - The node v, the operation corresponding to the expression being processed (e.g. And for *and*, etc.), and replication dimensions dv, is added. The node has n data inputs: for $1 \leq i \leq n$, v^i, with identity coordinate mapping;
   - The nodes o^i for $1 \leq i \leq n$, with the operation Or, replication dimensions dv, and a single input o, with identity dimension mapping, are added.

Theorem 1. For any $e \in \mathbf{Expr}$, $dv \in \mathcal{P}(\mathbf{Dim})$, DG g (corresponding to the DGR gr), $vartonodes \in \mathbf{Var} \to (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})$, $cn \in \mathbf{Lab}$, the configuration $inp, rand \vdash \langle ts, s \rangle$, and the graph configuration $\rho^{init}, \phi^{init}, \psi^{init}$ satisfying the following conditions:

- $vartonodes$ defined for all $FV(e)$,
- $vartonodes\, x = x^o, x^a, x^v$, such that:
  - The nodes x^o (operation Or), x^a (operation And), and x^v (any operation), belong to the gr;
  - The nodes x^o, x^a, and x^v have same number of coordinates in each replication dimension;
  - For any coordinate vector cv, corresponding to the dimensions of the x^o, and $\langle \rho', \phi', \psi' \rangle = Eval^g(\langle \rho^{init}[x^o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle)$, the $\rho'[x^a.cv] = \mathsf{true}$, $\rho'[x^v.cv] = s\, x\, cv$, and $\rho'[c^n.cv] = \mathsf{true}$
- $rand\, l = \sigma \Rightarrow \psi[l] = \sigma$
- $inp\, l = \sigma \Rightarrow \phi[l] = \sigma$

and $gr', o, a, v = EtoDG[e]\, dv\, gr\, vartonodes\, cn$ the following holds (g' is the DG corresponding to the DGR gr'): For any coordinate vector cv, corresponding to the replication dimensions dv, and $\langle \rho'', \phi'', \psi'' \rangle = Eval^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle)$, the $\rho''[a.cv] = \mathsf{true}$, and $\rho''[v.cv] = inp, rand \vdash \mathcal{A}[\![e]\!]\, cv\, s$.
Proof. The proof is done by induction on the syntactic structure of the e. First we show that in the base cases (when the translation is made in one step) the theorem holds. Then, assuming that the theorem holds for translating the subexpressions $e_1, \ldots, e_n$ of e, we show that the theorem also holds for the construction of e.

In course of the proof we assume that the $EtoDG$ parameters are fixed. Also, we take some fixed coordinate vector cv, corresponding to the replication dimensions dv.

The bases cases are:

1. e is x.

   Let $x^o, x^a, x^v = vartonodes\, x$ (the function is defined for $x \in FV(e)$). Let dv' be the replication dimensions of the node x^o. Two cases are possible:

   (a) $dv' = dv$.
   Let $\langle \rho', \phi', \psi' \rangle = Fix(Eval^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   i. $\rho'[x^o.cv] = \mathsf{true}$ (by the definition of f^{x^o});

ii. $\rho'[x^a.cv] = \mathsf{true}$, $\rho'[x^v.cv] = s\,x\,cv$, $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g, *vartonodes*, and cn);

iii. $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a)

iv. $\rho'[v.cv] = \rho'[x^v.cv] = s\,x\,cv$ (by the definition of f^v)

But, by definition of $\mathcal{A}$, $inp, rand \vdash \mathcal{A}[\![x]\!]cv\,s = s\,x\,cv$. So, for this sub-case the required result holds.

(b) $dv' < dv$.
Let cv' be the subset of cv coordinates, according to the subset of dimensions dv'). Then, as in the previous sub-case, let $\langle \rho', \phi', \psi' \rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

i. $\rho'[lo.cv'] = \mathsf{true}$ (by the definition of f^{lo}),

ii. $\rho'[x^o.cv'] = \mathsf{true}$ (by the definition of f^{x^o}),

iii. $\rho'[x^a.cv'] = \mathsf{true}$, $\rho'[x^v.cv'] = s\,x\,cv'$, $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g, *vartonodes*, and cn);

iv. $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a)

v. $\rho'[v.cv] = \rho'[x^v.cv] = s\,x\,cv'$ (by the definition of f^v)

Then note that, by the definition of s, the
$(s\,x\,cv^1 = \sigma^1 \wedge (cv^2\ \mathit{includes}\ cv^1)) \implies s\,x\,cv^2 = \sigma^1$. Therefore, the $s\,x\,cv' = s\,x\,cv$. Finally, by the definition of $\mathcal{A}$, $inp, rand \vdash \mathcal{A}[\![x]\!]\,cv\,s = s\,x\,cv$. This completes the proof for $e = x$.

2. e is an expression with no arguments and not using the random coins. The expressions $secret^l$, $receive^l$, and n fall into this category.

   Let $\langle \rho', \phi', \psi' \rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   (a) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g).

   (b) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a)

   (c) $\chi[v.cv]$ is defined (by f^v), where χ is ψ if the expression translated is $secret^l$, ϕ if the expression is $receive^l$, or ρ if the expression is n.

   As the v was set to be equal to the label of the expression l, the $\psi[v.cv] = rand\,l\,cv$ (if *secret* was translated), or $\phi[v.cv] = inp\,l\,cv$ (if *receive* was translated). If e is n, the above property is not significant for the proof. In either case, the $inp, rand \vdash \mathcal{A}[\![e]\!]\,cv\,s$ is equal to $\chi[v.cv]$.

3. e is an expression with no arguments, using the random coins. The expressions $nonce^l$, $keypair^l$, $symkey^l$, and $svkpair^l$ fall into this category. Let $\langle \rho', \phi', \psi' \rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

(a) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g);

(b) $\rho'[r^o.cv] = \mathsf{true}$ (by the definition of f^{r^o});

(c) $\rho'[r^a.cv] = \mathsf{true}$ (by the definition of f^{r^a});

(d) $\rho'[r^v.cv] = \psi[r^v]$ (by the definition of f^{r^v});

(e) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a);

(f) $\rho'[v.cv] = op(\psi[r^v.cv])$ (by the definition of f^v), where op is $nonce^l$, $keypair^l$, $symkey^l$, or $svkpair^l$.

As the r^v is equal to the label of the expression, the $op(\psi[r^v.cv]) = op(rand\ l\ cv)$ (where op is the operation in question). Then, by definition of $\mathcal{A}$, $inp, rand \vdash \mathcal{A}[\![op^l]\!]\ cv\ s = op(rand\ l\ cv)$, which what the $\rho'[v.cv]$ is equal to.

The "induction step" cases are:

1. e is an arithmetic expression taking one or more arguments, and not using the random coins. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $pubkey(a)$, $verkey(a)$, $getmsg(a)$, $proj_{n,n'}(a)$ (1 argument), $pubdec(a_1, a_2)$, $symdec(a_1, a_2)$ (2 arguments), and $tuple_m(a_1, \ldots, a_m)$ (m arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i.

   Let $gr^0 = gr$. We assume the theorem holds for each i, $1 \leq i \leq n$, and $gr^i, o^i, a^i, v^i = EtoDG[a_i]\ dv\ gr^{i-1}\ vartonodes\ cn$. Note that the induction step is made for the subexpressions independently, since, when translating the i-th subexpression, the nodes corresponding to the $i-1$-th are not used.

   Let $\langle \rho', \phi', \psi' \rangle = Fix(Eval^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   (a) $\forall i, 1 \leq i \leq n. \rho'[o^i.cv] = \mathsf{true}$ (by the definition of f^{o^i}),

   (b) $\forall i, 1 \leq i \leq n. \rho'[a^i.cv] = \mathsf{true}$, $\rho'[v^i.cv] = inp, rand \vdash \mathcal{A}[\![a_i]\!]\ cv\ s$ (by the induction hypothesis);

   (c) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g^1);

   (d) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a);

   (e) $\rho'[v.cv] = op(\rho'[v^1.cv], \ldots, \rho'[v^n.cv]) =$
   $= op(inp, rand \vdash \mathcal{A}[\![a_1]\!]\ cv\ s, \ldots, inp, rand \vdash \mathcal{A}[\![a_n]\!]\ cv\ s)$ — by the definition of f^v (op is the expression translated.)

But, by definition of $\mathcal{A}$, $inp, rand \vdash \mathcal{A}[\![op(a_1, \ldots, a_n)]\!]\, cv\, s$ is equal to $op(inp, rand \vdash \mathcal{A}[\![a_1]\!]\, cv\, s, \ldots, inp, rand \vdash \mathcal{A}[\![a_n]\!]\, cv\, s)$, which is what we are required to prove.

2. e is an arithmetic expression taking one or more parameters and using random coins. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $pubenc^l(a_1, a_2)$, $symenc^l(a_1, a_2)$, and $sig^l(a_1, a_2)$.

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i.

   Let $gr^0 = gr$. We assume the theorem holds for each i, $1 \leq i \leq n$, and $gr^i, o^i, a^i, v^i = EtoDG[a_i]\; dv\; gr^{i-1}\; vartonodes\; cn$. Note that the induction step is made for the subexpressions independently, since, when translating the i-th subexpression, the nodes corresponding to the $i-1$-th are not used.

   Let $\langle \rho', \phi', \psi' \rangle = Fix(Eval^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   (a) $\forall i, 1 \leq i \leq n. \rho'[o^i.cv] = \mathsf{true}$ (by the definition of f^{o^i}),
   (b) $\rho'[r^o.cv] = \mathsf{true}$ (by the definition of f^{r^o}),
   (c) $\forall i, 1 \leq i \leq n. \rho'[a^i.cv] = \mathsf{true}$, $\rho'[v^i.cv] = inp, rand \vdash \mathcal{A}[\![a_i]\!]\, cv\, s$ (by the induction hypothesis);
   (d) $\rho'[r^a.cv] = \mathsf{true}$ (by the definition of f^{r^a});
   (e) $\rho'[r^v.cv] = \psi'[r^v.cv]$ (by the definition of f^{r^v});
   (f) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g^1);
   (g) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a);
   (h) $\rho'[v.cv] = op(\psi[r^v.cv], \rho'[v^1.cv], \ldots, \rho'[v^n.cv]) =$
   $= op(rand\; l\; cv, inp, rand \vdash \mathcal{A}[\![a_1]\!]\, cv\, s, \ldots, inp, rand \vdash \mathcal{A}[\![a_n]\!]\, cv\, s)$
   — by the definition of f^v (op is the expression translated.)

   But, by definition of $\mathcal{A}$, $inp, rand \vdash \mathcal{A}[\![op(a_1, \ldots, a_n)]\!]\, cv\, s$ is equal to $op(rand\; l\; cv, inp, rand \vdash \mathcal{A}[\![a_1]\!]\, cv\, s, \ldots, inp, rand \vdash \mathcal{A}[\![a_n]\!]\, cv\, s)$, which is what we are required to prove.

3. e is a boolean expression taking arithmetic arguments. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $isok(a)$ (1 argument), $a_1 = a_2$, $a_1 \neq a_2$, $testsig(a_1, a_2)$ (2 arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i.

Let $gr^0 = gr$. We assume the theorem holds for each i, $1 \leq i \leq n$, and $gr^i, o^i, a^i, v^i = \mathit{EtoDG}[a_i]\ dv\ gr^{i-1}\ \mathit{vartonodes}\ cn$. Note that the induction step is made for the subexpressions independently, since, when translating the i-th subexpression, the nodes corresponding to the $i-1$-th are not used.

Let $\langle \rho', \phi', \psi' \rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

(a) $\forall i, 1 \leq i \leq n. \rho'[o^i.cv] = \mathsf{true}$ (by the definition of f^{o^i}),

(b) $\forall i, 1 \leq i \leq n. \rho'[a^i.cv] = \mathsf{true}$, $\rho'[v^i.cv] = \mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a_i]\!]\ cv\ s$ (by the induction hypothesis);

(c) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g^1);

(d) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a);

(e) $\rho'[v'.cv] = op(\rho'[v^1.cv], \ldots, \rho'[v^n.cv]) =$
$= op(\mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a_1]\!]\ cv\ s, \ldots, \mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a_n]\!]\ cv\ s)$ — by the definition of $f^{v'}$ (op is the expression translated);

(f) $\rho'[v.cv] = \mathsf{true} \wedge \rho'[v'.cv] =$
$= op(\mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a_1]\!]\ cv\ s, \ldots, \mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a_n]\!]\ cv\ s)$ — by the definition of f^v.

The value of the $\rho'[l^v.cv]$ is defined in the same way as $\mathit{inp}, \mathit{rand} \vdash \mathcal{B}[op(a_1, \ldots, a_n)]\ cv\ s$

4. e is a boolean expression taking boolean arguments. Out of the expressions we supplied in the protocol language, the following operations fall into this group: $\mathit{and}(b_1, \ldots, b_m)$ and $\mathit{or}(b_1, \ldots, b_m)$ (m arguments).

   Let the number of the expression arguments be n, and the i-th argument (for $1 \leq i \leq n$) be a_i.

   Let $gr^0 = gr$. We assume the theorem holds for each i, $1 \leq i \leq n$, and $gr^i, o^i, a^i, v^i = \mathit{EtoDG}[b_i]\ dv\ gr^{i-1}\ \mathit{vartonodes}\ cn$. Note that the induction step is made for the subexpressions independently, since, when translating the i-th subexpression, the nodes corresponding to the $i-1$-th are not used.

   Let $\langle \rho', \phi', \psi' \rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   (a) $\forall i, 1 \leq i \leq n. \rho'[o^i.cv] = \mathsf{true}$ (by the definition of f^{o^i}),

   (b) $\forall i, 1 \leq i \leq n. \rho'[a^i.cv] = \mathsf{true}$, $\rho'[v^i.cv] = \mathit{inp}, \mathit{rand} \vdash \mathcal{B}[b_i]\ cv\ s$ (by the induction hypothesis);

   (c) $\rho'[cn.cv] = \mathsf{true}$ (by the requirements we put on g^1);

(d) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a);

(e) $\rho'[v.cv] = op(\rho'[v^1.cv], \dots, \rho'[v^n.cv]) =$
$= op(inp, rand \vdash \mathcal{B}[b_1]\ cv\ s, \dots, inp, rand \vdash \mathcal{B}[b_n]\ cv\ s)$ — by the definition of f^v (op is the expression translated.)

The value of the $\rho'[l^v.cv]$ is defined in the same way as $inp, rand \vdash \mathcal{B}[op(b_1, \dots, b_n)]\ cv\ s$

Having shown that the theorem holds in all the base and induction cases, we have it proved. □

5.2 *AStoDG* — translation from *AS* to DGR

Formally, the function has type:

$$\begin{aligned} AStoDG \quad &: \quad \mathbf{AStmt} \\ &\rightarrow \quad (\mathcal{P}(\mathbf{Dim}) \times \mathbf{GR} \times (\mathbf{Var} \rightarrow (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})) \times \\ &\quad\quad \times \mathbf{Lab} \times \{\mathbf{Lab}\} \\ &\rightarrow \quad (\mathbf{GR} \times \mathbf{Lab} \times (\mathbf{Var} \rightarrow (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})))) \end{aligned}$$

The arguments to the *AStoDG* function are:

- $AS \in \mathbf{AStmt}$: statement to translate to the DGR;
- $dv \in \mathcal{P}(\mathbf{Dim})$: function specifying the replication dimensions the atomic statement should have a coordinate in;
- $gr \in \mathbf{GR}$: DGR, containing the operations for calculating the variables which may be used in AS, the continuation and request nodes;
- $vartonodes \in \mathbf{Var} \rightarrow (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})$: Function returning for each of the variables that many be used in AS the labels of Or-node, And-node, and *val*-node.
- $cndown \in \mathbf{Lab}$: the continuation node. The dependency, the value of which should be equal to true, in order for the statement to be executed; this parameter is optional;
- $cnups \in \{\mathbf{Lab}\}$: the request node(s). If any of these dependencies is set to true, the evaluation of the DG corresponding to DGR g is triggered.

Evaluation of $AStoDG[AS]\ dv\ gr\ vartonodes\ cndown\ cnups$ results in the gr', the labels o (node to be set to true to initiate the atomic statement evaluation), a (when the atomic statement evaluation is complete, it is set to true), the mapping $vartonodes'$ (reflecting the variable mapping after the statement is executed), and proceeds in the following way, depending on what AS is:

1. AS is $x := a$. Let $gr^e, a^o, a^a, a^v = EtoDG[a]$ dv gr $vartonodes$ $cndown$. First, each of the nodes belonging to $cnups$ is replaced with a new one. The following procedure is performed for each $cnup \in cnups$ (let the DGR with all the $cnup$ nodes processed be gr^s):
   (a) If the $cnup$ has same replication dimensions as dv, then:
      - The node $cnup$ (with n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
      - The node $cnup$, Or operation, and replication dimensions dv is added. The node has $n+1$ inputs:
         – For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i);
         – The $n+1$-th input is a^o, with the identity coordinate mapping.

   (b) If $cnup$ (let its replication dimensions be dv') has less coordinates than specified by dv, then:
      - The node $cnup$ (with n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
      - The node lo, the operation Longor, and replication dimensions dv', is added. The input is a^o, with identity coordinate mapping. All the dv coordinates, not existing in dv', are contracted;
      - The node $cnup$, Or operation, the same replication dimensions as $cnup$ had, is added. The node has $n+1$ inputs:
         – For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
         – The $n+1$-th input is lo, with the identity coordinate mapping.

   The result of the translation is $gr', a^o, a^a, vartonodes[x \mapsto a^o, a^a, a^v]$.

2. AS is $send^l$ a.

   Let $gr^e, a^o, a^a, a^v = EtoDG[a]$ dv gr $vartonodes$ $cndown$. First, each of the nodes belonging to $cnups$ is replaced with a new one. The following procedure is performed for each $cnup \in cnups$ (let the DGR with all the $cnup$ nodes processed be gr^s):
   (a) If the $cnup$ has same replication dimensions as dv, then:
      - The node $cnup$ (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
      - The node $cnup$, Or operation, replication dimensions dv, is added. The node has $n+1$ inputs:

  - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
  - The $n+1$-th input is a^o, with the identity coordinate mapping;
- The node a^o (having no inputs) is removed;
- The node a^o, Or operation, and replication dimensions dv, is added. The node input is req, with identity coordinate mapping;
- The node req, operation Req, and replication dimensions dv, is added;
- The node a, And operation, and replication dimensions dv, is added. The node inputs are:
  - req, with identity coordinate mapping;
  - a^a, with identity coordinate mapping;
- The node s, operation Send, and replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The data input is a^v, with identity coordinate mapping.

(b) If $cnup$ (let its replication dimensions be dv') has less replication dimensions than dv, then:

- The node $cnup$ (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
- The node lo, the operation Longor, replication dimensions dv', the input a^o, with identity coordinate mapping, is added. All the dv coordinates, not existing in dv', are contracted;
- The node $cnup$, Or operation, replication dimension dv', is added. The node has $n+1$ inputs:
  - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
  - The $n+1$-th input is lo, with the identity coordinate mapping;
- The node a^o (having no inputs) is removed;
- The node req, operation Req, and replication dimensions dv, is added;
- The node a^o, Or operation, replication dimensions dv, is added. The node input is req, with identity coordinate mapping;
- The node a, And operation, replication dimensions dv, is added. The node inputs are:
  - req, with identity coordinate mapping;
  - a^a, with identity coordinate mapping;

- The node s, Send operation, replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The data input is a^v, with identity coordinate mapping.

The result of the translation is $grr^s, a^o, a^a, vartonodes$.

3. AS is $begin^l\ a$, or $end^l\ a$.

   Let $gr^e, a^o, a^a, a^v = EtoDG[a]\ dv\ gr\ vartonodes\ cndown$. In the below description, *begin* is used as an example; *end* is processed in the same way. First, each of the nodes belonging to *cnups* is replaced with a new one. The following procedure is performed for each $cnup \in cnups$ (let the DGR with all the *cnup* nodes processed be gr^s):

   (a) If the *cnup* has same replication dimensions as dv, then:
   - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
   - The node *cnup*, Or operation, replication dimensions dv, is added. The node has $n + 1$ inputs:
     - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
     - The $n+1$-th input is a^o, with the identity coordinate mapping;
   - The node a, And operation, replication dimensions dv, is added. The node inputs are:
     - a^o, with identity coordinate mapping;
     - a^a, with identity coordinate mapping;
   - The node s, Begin operation, replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The data input is a^v, with identity coordinate mapping.

   (b) If *cnup* (let its replication dimensions be dv') has less replication dimensions than dv, then:
   - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
   - The node lo, Longor operation, replication dimensions dv', the input a^o, with identity coordinate mapping. All the dv coordinates, not existing in dv', are contracted;
   - The node *cnup*, Or operation, replication dimensions dv', is added. The node has $n + 1$ inputs:
     - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;

    - The $n+1$-th input is lo, with the identity coordinate mapping;
- The node a, operation And, replication dimensions dv, is added. The node inputs are:
    - a^o, with identity coordinate mapping;
    - a^a, with identity coordinate mapping;
- The node s, Begin operation, replication dimensions dv, is added. The control input is a, with identity coordinate mapping. The data input is a^v, with identity coordinate mapping.

The result of the translation is $gr^s, a^o, a^a, vartonodes$.

We say that the gr, $cnups$, $cndown$, $vartonodes$ captures the protocol state s in the environment inp, $rand$, if for some coordinate vector cv^{cnup}, the evaluation of the DG g (corresponding to the DGR gr) in the DG state $\langle \rho^{init}, \phi^{init}, \psi^{init} \rangle$, defined as following:

- ρ^{init}, mapping some $cnup \in cnups.cv^{cnup}$ to true, and everything else to $\perp/\mathsf{false}$;
- ϕ^{init}, such that for all l, cv^l, $inp\,l\,cv^l = \phi^{init}[l.cv^l]$;
- ψ^{init}, such that for all l, cv^l, $rand\,l\,cv^l = \psi^{init}[l.cv^l]$;

results in the DG state $\langle \rho^{final}, \phi^{final}, \psi^{final} \rangle$, such that $\rho^{final}[cndown] = \mathsf{true}$, and if (for any x and cv) the $s\,x\,cv = \sigma$, then:

- $vartonodes\,x\,cv = x^a, x^o, x^v$;
- $\rho^{final}[x^o] = \rho^{final}[x^a] = \mathsf{true}$;
- $\rho^{final}[x^v] = \sigma$.

Theorem 2. Let gr, $cnups$, $cndown$, $vartonodes$ capture the state s. For any atomic statement AS, and any replication dimensions dv, the
$gr', cnup', cndown', vartonodes' = AStoDG[AS]\;dv\;vartonodes\;cnup, cndown$
captures (for any cv') the state $s' = \mathcal{AS}[\![AS]\!]cv's$.
Proof. The theorem validity can be checked by examining all the possible steps performed in the $AStoDG$ (which is fairly trivial, as the function is not recursive).

5.3 *StoDG* — translation from S to DGR

Formally, the function has type:

$$\begin{aligned} \mathit{StoDG} \;:\; & \mathbf{Stmt} \\ \to\; & (\mathcal{P}(\mathbf{Dim}) \times \mathbf{GR} \times (\mathbf{Var} \to (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})) \times \\ & \times \mathbf{Lab} \times \{\mathbf{Lab}\} \\ \to\; & (\mathbf{GR} \times \mathbf{Lab})) \end{aligned}$$

The arguments of the *StoDG* function are:

- $S \in \mathbf{Stmt}$: statement to translate to the DGR;
- $dv \in \mathcal{P}(\mathbf{Dim})$: function specifying the replication dimensions the statement should have a coordinate in;
- $gr \in \mathbf{GR}$: DGR, containing the operations for calculating the variables which may be used in S, the continuation and request nodes;
- $vartonodes \in \mathbf{Var} \to (\mathbf{Lab} \times \mathbf{Lab} \times \mathbf{Lab})$: Function returning for each of the variables that many be used in S the labels of Or-node, And-node, and *val*-node.
- $cndown \in \mathbf{Lab}$: the continuation node. The dependency, the value of which should be equal to true, in order for the statement to be executed; this parameter is optional;
- $cnup \in \{\mathbf{Lab}\}$: the request node. Setting any of the dependencies in this set to true, triggers the evaluation of the DG corresponding to DGR g; this parameter is optional.

Evaluation of $\mathit{StoDG}[S]\ dv\ gr\ vartonodes\ cndown\ cnups$ results in the gr', the set of labels o, and proceeds in the following way, depending on what the S is:

1. S is *stop*. In this case the gr' is equal to gr and o equal to $cnups$.
2. S is $AS; S_1$.

   Let
   $gr^{as}, as^o, as^a, vartonodes' = \mathit{AStoDG}[AS]\ dv\ gr\ vartonodes\ cndown\ cnup$.
   The result of the translation is
   $gr', o = \mathit{StoDG}[S_1]\ dv\ gr^{as}\ vartonodes'\ as^a\ as^o$.
3. S is $par_n(S_1, \ldots, S_n)$.

The result of the translation is $gr', \cup_{i=1}^{n}\{o^i\}$, defined as following:

$$\begin{array}{rcl} gr^1, o^1 & = & StoDG[S_1]\ dv\ gr\ vartonodes\ cndown\ cnup \\ \ldots & & \\ gr', o^n & = & StoDG[S_n]\ dv\ gr^{n-1}\ vartonodes\ cndown\ cnup \end{array}$$

4. S is $rep(d, S_1)$.

   The result of the translation is gr', o, defined as following:

   $$gr', o \ = \ StoDG[S_1]\ dv'\ gr\ vartonodes\ cndown\ cnup$$

   The dv' is obtained from dv by adding a coordinate to the dimension d.

5. S is *if* b *then* S_1.

   Let $gr^e, b^o, b^a, b^v = EtoDG[b]\ dv\ gr\ vartonodes\ cndown$. First, each of the nodes belonging to *cnups* is replaced with a new one. The following procedure is performed for each $cnup \in cnups$ (let the DGR with all the *cnup* nodes processed be gr^s):

   (a) If the *cnup* has same replication dimensions as *dv*, then:
   - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
   - The node *cnup*, Or operation, replication dimensions *dv* is added. The node has $n+1$ inputs:
     - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
     - The $n+1$-th input is b^o, with the identity coordinate mapping.

   (b) If *cnup* (let its replication dimensions be dv') has less replication dimensions than *dv*, then:
   - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
   - The node *lo*, Longor operation, replication dimensions dv', the input b^o, with identity coordinate mapping, is added. All the *dv* coordinates, not existing in dv', are contracted;
   - The node *cnup*, Or operation, replication dimensions dv', is added. The node has $n+1$ inputs:
     - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;

      - The $n+1$-th input is lo, with the identity coordinate mapping.

The results of the translation is
$gr', o' = StoDG[S_1]\ dv\ gr^s\ vartonodes\ b^a\ b^o$

6. S is *iff* b *then* AS_1 *finally* S_1.

   Let $gr^e, b^o, b^a, b^v = EtoDG[b]\ dv\ gr\ vartonodes\ cndown$. First, each of the nodes belonging to *cnups* is replaced with a new one. The following procedure is performed for each $cnup \in cnups$ (let the DGR with all the *cnup* nodes processed be gr^s):

   (a) If the *cnup* has same replication dimensions as *dv*, then:
      - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
      - The node *cnup*, Or operation, replication dimensions *dv* is added. The node has $n+1$ inputs:
        - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
        - The $n+1$-th input is b^o, with the identity coordinate mapping.

   (b) If *cnup* (let its replication dimensions be dv') has less replication dimensions than *dv*, then:
      - The node *cnup* (having n inputs; let the i-th input be $cnup^i$, with the coordinate mapping λ^i) is removed;
      - The node *lo*, Longor operation, replication dimensions dv', the input b^o, with identity coordinate mapping, is added. All the *dv* coordinates, not existing in dv', are contracted;
      - The node *cnup*, Or operation, replication dimensions dv', is added. The node has $n+1$ inputs:
        - For $1 \leq i \leq n$ the i-th input is $cnup^i$, with the coordinate mapping λ^i;
        - The $n+1$-th input is *lo*, with the identity coordinate mapping.

   The results of the translation is gr^2, o^2, defined as following:

$$gr^1, o^1, a^1, vartonodes^1 = AStoDG[AS_1]\ dv\ gr^s\ vartonodes\ b^a\ b^o$$
$$gr^2, o^2 = StoDG[S_1]\ dv\ gr^1\ vartonodes^1\ cndown\ \{b^o, o^1\}$$

Theorem 3. Given the:

- protocol state s and environment $inp, rand$,
- DG g, corresponding to the DGR gr,
- nodes $cnup$, $cndown$, function $vartonodes$,
- DG state $\langle \rho^{init}, \phi^{init}, \psi^{init} \rangle$,

such that for $\langle \rho^1, \phi^1, \psi^1 \rangle = Eval^g(\langle \rho^{init}[cnup \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle)$, the $(s\, x\, cv = \sigma) \Rightarrow \rho^1[x^a.cv] = \mathsf{true} \wedge \rho^1[x^v.cv] = \sigma$ $((x^o, x^a, x^v) = vartonodes\, x)$, the DG g', corresponding to DGR gr', $cnup'$, constructed by $StoDG[S]\ dv\ gr\ vartonodes\ cndown\ cnup$, will capture the adversary view, generated by the statement S, executed in state s.

Specifically, for some cv, if $inp, rand \vdash \mathcal{S}\langle (S, cv), s \rangle \Rightarrow^* \langle (send^l\, a\, S', cv') \cup ts, s'' \rangle$ and $inp, rand \vdash \mathcal{A}[\![a]\!]\ cv'\ s'' = \sigma$, then there exist two nodes req (Req operation) and val (Send operation), such that for $\langle \rho^2, \phi^2, \psi^2 \rangle = Eval^{g'}(\langle \rho^{init}[req.cv' \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle)$, the $\rho^2[val.cv'] = \sigma$.

Note that during the execution of the protocol, specified in the procedural language, the choices of adversary (feeding the participants with the values received from the network and the participants scheduling) are made by selecting the next thread to execute. In case of DG the adversary controls the execution by setting that values of Receive- and Req-nodes.

Proof. The proof is constructed by induction on the length of the derivation sequence $\langle \{S, cv\}, s \rangle \Rightarrow^* \langle \{send^l\, a\, S', cv'\} \cup ts, s'' \rangle$. Note that the case S is *stop* is not considered in the proof, as *stop* is the final statement in the protocol (so there can be no *send*-statement (the property of which we prove) after it). The possible cases are:

1. Base case

   We show that the theorem holds for all the derivation sequences of length 0. The derivation sequence $\langle \{S, cv\}, s \rangle \Rightarrow^* \langle \{send^l\ a\ S', cv'\} \cup ts, s'' \rangle$ has the length 0 if and only if $S = send^l\ a\ S'$.

   Let $\langle \rho', \phi', \psi' \rangle = Fix(Eval^{g^s}(\langle \rho^{init}[req.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init} \rangle))$. We observe that:

   (a) The node lo is present only if the translated statement has more coordinates than $cnup$; $\rho'[lo.cv] = \mathsf{true}$(by definition of f^{lo})

   (b) $\rho'[cnup.cv] = \mathsf{true}$ (by definition of f^{cnup}),

   (c) $\rho'[a^o.cv] = \mathsf{true}$ (by the definition of f^{a^o}),

   (d) $\rho'[a^a.cv] = \mathsf{true}$ (by the Theorem 1),

   (e) $\rho'[a^v.cv] = inp, rand \vdash \mathcal{A}[\![a]\!]\ cv\ s$ (by Theorem 1),

   (f) $\rho'[a.cv] = \mathsf{true}$ (by the definition of f^a),

   (g) $\rho'[s.cv] = \mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a]\!]\, cv\, s$ (by the definition of f^s).

   The last equality ($\rho'[s.cv] = \mathit{inp}, \mathit{rand} \vdash \mathcal{A}[\![a]\!]\, cv\, s$) is what is required to prove.

2. Induction step

   Assuming the theorem holds for all the derivation sequences of length k_0, we show that it also holds for the sequences of length $k_0 + 1$.

   The derivations in sequence of length $k_0 + 1$

$$\langle \{S, cv\}, s\rangle \Rightarrow^{k_0+1} \langle (\mathit{send}^l\ a\, S', cv') \cup ts, s''\rangle$$

   starts with configuration rewriting according to one of the semantic rules, depending on what the S is, so the aforementioned sequence can be rewritten as:

$$\langle (S, cv), s\rangle \Rightarrow \langle ts, s'\rangle \Rightarrow^{k_0} \langle (\mathit{send}^l\ a\, S', cv') \cup ts, s''\rangle$$

   For each case (depending on what the first statement is), we show that the intermediate graph constructed inside the *StoDG* and passed during the recursive call to *StoDG* (for a statement, corresponding to derivation sequence of length of maximum k_0) satisfies the premises of the theorem (so, by induction hypothesis, it holds).

   The possible cases are:

   (a) S is $AS_1; S_1$.

   In this case the derivation sequence is

$$\begin{array}{rcl} \mathit{inp}, \mathit{rand} \vdash \langle (AS; S_1, cv), s\rangle & \Rightarrow & \langle (S_1, cv), s'\rangle \\ & \Rightarrow^{k_0} & \langle (\mathit{send}^l\ a\, S', cv') \cup ts, s''\rangle \end{array}$$

   Per Theorem 2, the gr^{as}, produced by the *AStoDG*, captures every variable, defined in the state s', therefore, satisfying all the requirements put on the arguments to the *StoDG* function, so for the DG g^1, o, the statement $S_1\ dv$, and the state s' the theorem holds by induction hypothesis.

   (b) S is $\mathit{par}_n(S_1, \ldots, S_n)$.

   In this case the derivation sequence is

$$\begin{array}{rl} \mathit{inp}, \mathit{rand} \vdash \langle (\mathit{par}_n(S_1, \ldots, S_n), cv), s\rangle \Rightarrow & \langle \cup_{i=1}^n \{(S_i, cv)\}, s\rangle \\ \Rightarrow^{k_0} & \langle \{\mathit{send}^l\ a\, S', cv'\} \cup ts, s''\rangle \end{array}$$

For each of the DGs g^i obtained by translating the i-th statement, the theorem holds by induction hypothesis (as the derivation sequence for each of the statements can contain at most k_0 steps).

(c) S is $rep(d, S_1)$.
In this case the derivation sequence is

$$\begin{aligned} inp, rand \vdash \langle (rep(d, S_1), cv), s\rangle \quad &\Rightarrow \quad \langle \bigcup_{j\in\mathbf{N}} \{(S_1, cv[d \mapsto j])\}, s\rangle \\ &\Rightarrow^{k_0} \quad \langle (send^l\ a\ S', cv') \cup ts, s''\rangle \end{aligned}$$

For the DG g^1, obtained by translating the statement S_1, the theorem holds by induction hypothesis (as the derivation sequence for it can contain at most k_0 steps).

(d) S is $if\ b\ then\ S_1$.
In this case the derivation sequence is either

$$\begin{aligned} inp, rand \vdash \langle (if\ b\ then\ S_1\ cv), s\rangle \Rightarrow \\ \Rightarrow \langle (S_1, cv), s\rangle \Rightarrow^{k_0} \langle (send^l\ a\ S', cv') \cup ts, s''\rangle \end{aligned}$$

if $\mathcal{B}[\![b]\!]\ cv\ s = \mathsf{true}$, or

$$inp, rand \vdash \langle (if\ b\ then\ S_1\ cv), s\rangle \Rightarrow \langle \emptyset, s\rangle$$

if $\mathcal{B}[\![b]\!]\ cv\ s = \mathsf{false}$.
Let $\langle \rho', \phi', \psi'\rangle = Fix(Eval^{g'}(\langle \rho^{init}[b^o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init}\rangle))$. We observe that:

i. The operation lo is present only if the translated statement has more coordinates than $cnup$; $\rho'[lo.cv] = \mathsf{true}$ (by definition of f^{lo})
ii. $\rho'[cnup.cv] = \mathsf{true}$ (by definition of f^{cnup}),
iii. $\rho'[l^{ba}.cv] = \mathsf{true}$ (by Theorem 1),
iv. $\rho'[l^{bv}.cv] = inp, rand \vdash \mathcal{B}[b]\ cv\ s$ (by Theorem 1),

The induction hypothesis can be applied to the $gr', o' = StoDG[S_1]\ dv\ gr^s\ vartonodes\ b^a\ b^o$, as it is evaluated from the state s in at most k_0 derivations. Note that the b^v (the semantics of which are equal to $inp, rand \vdash \mathcal{B}[\![b]\!]\ cv\ s$) defines whether the nodes of the graph, corresponding to the S_1 are evaluated or not (exactly as in the protocol semantics the corresponding derivation rules is chosen). The second case (if the S_1 is not evaluated is trivial — as the derivation sequence is completed in one step).

(e) S is *iff b then* AS_1 *finally* S_1.

In this case the derivation sequence is either

$$\begin{aligned} &inp, rand \vdash \langle (\mathit{iff}\ b\ \mathit{then}\ AS_1\ \mathit{finally}\ S_1, cv), s\rangle \Rightarrow \\ &\Rightarrow \langle (AS_1; S_1, cv), s\rangle \Rightarrow^{k_0} \langle (\mathit{send}^l\ a\ S', cv') \cup ts, s''\rangle \end{aligned}$$

if $\mathcal{B}[\![b]\!]\ cv\ s = \mathsf{true}$, or

$$\begin{aligned} &inp, rand \vdash \langle (\mathit{iff}\ b\ \mathit{then}\ AS_1\ \mathit{finally}\ S_1, cv), s\rangle \Rightarrow \\ &\Rightarrow \langle (S_1, cv), s\rangle \Rightarrow^{k_0} \langle (\mathit{send}^l\ a\ S', cv') \cup ts, s''\rangle \end{aligned}$$

if $\mathcal{B}[\![b]\!]\ cv\ s = \mathsf{false}$.

Let $\langle \rho', \phi', \psi'\rangle = \mathit{Fix}(\mathit{Eval}^{g'}(\langle \rho^{init}[b^o.cv \mapsto \mathsf{true}], \phi^{init}, \psi^{init}\rangle))$. We observe that:

i. The operation lo is present only if the translated statement has more coordinates than $cnup$; $\rho'[lo.cv] = \mathsf{true}$ (by definition of f^{lo})

ii. $\rho'[cnup.cv] = \mathsf{true}$ (by definition of f^{cnup}),

iii. $\rho'[l^{ba}.cv] = \mathsf{true}$ (by Theorem 1),

iv. $\rho'[l^{bv}.cv] = inp, rand \vdash \mathcal{B}[\![b]\!]\ cv\ s$ (by Theorem 1),

Per Theorem 2, the gr^1, and *cnup* / *cndown* nodes produced by the *AStoDG*, satisfy the requirements put on the arguments to the *StoDG* function. The induction hypothesis can be applied to the $gr^2, o^2 = \mathit{StoDG}[S_1]\ dv\ gr^1\ \mathit{vartonodes}^1\ \mathit{cndown}\ \{b^o, o^1\}$, as it is evaluated from the state s in at most k_0 derivations. Note that the b^v (the semantics of which are equal to $inp, rand \vdash \mathcal{B}[\![b]\!]\ cv\ s$) defines whether the nodes of the graph, corresponding to the AS_1 are evaluated or not (exactly as in the protocol semantics the corresponding derivation rules is chosen).

Having shown that for all the cases the required property holds, we've proved the Theorem 3. □

Theorem 4. The DG g, corresponding to the DGR gr, constructed with $\mathit{StoDG}[S]\ \emptyset\ gr^0\ \mathit{vartonodes}^0\ \emptyset, \emptyset$, where the gr^0 is an empty DGR, and $\mathit{vartonodes}^0$ has empty domain, captures the adversary view of the protocol S, as described in Theorem 3.

Proof. The validity of the theorem 4 follows from the Theorems 1, 2, and 3. □

6 Transforming the Dependency Graphs

Dependency graph transformation is replacement of one graph with another, the adversary view of which is computationally indistinguishable from the first one. We specify the transformation as an equivalence of two graph fragments (subgraphs with a certain interface to the graph which may contain it). Equivalence means that the two equivalent graph fragments, being executed, cannot be distinguished with non-negligible probability. The application of a transformation to the graph is replacing a subgraph, contained in it, with another equivalent subgraph. We show that the resulting graph, obtained by applying the transformation, is indistinguishable from the original one.

6.1 Dependency Graph Fragments

A dependency graph fragment (DGF) is a dependency graph without *Send*-, *Receive*- or *req*-nodes, but with extra input and output nodes (both boolean and bit-string). The operations defined on the graph fragments are listed in the Figure 10

$$\begin{array}{rcllll} \lambda^{gf}(n) & ::= & \lambda^{i}(n) & \mid & \text{InputB} & \mid \ \text{InputS} \\ & \mid & \text{OutputB}(B) & \mid & \text{OutputS}(X) & \end{array}$$

Figure 10: Dependency graph fragment operations

6.1.1 Semantics

The dependency graph fragment nodes are categorized in a way, similar to dependency graph nodes categorization. Let Lab^f be the set of all nodes of the dependency graph fragment. Let $\mathit{Lab}^f{}_*$ be the subset of Lab^f containing all nodes with operations RS or Secret. Let $\mathit{Lab}^f{}_\leftarrow \subseteq \mathit{Lab}$ be the set of all nodes with operations InputB or InputS. Let $\mathit{Lab}^f{}_\rightarrow$ be the set of all nodes with operations OutputB or OutputS. Let $\mathit{Lab}^f{}_\bullet = \mathit{Lab}^f \backslash (\mathit{Lab}^f{}_\leftarrow \cup \mathit{Lab}^f{}_*)$, i.e. $\mathit{Lab}^f{}_\bullet$ contains all nodes whose value is computed in the “usual” way, not making any use of some outside information.

A *Dependency graph fragment configuration* is a triple $\langle \rho^f, \phi^f, \psi^f \rangle \in \mathbf{Conf}$, where

- $\rho^f : \mathit{Lab}^f \rightarrow \mathbf{Val}$ gives the current value of each node;
- $\phi^f : \mathit{Lab}^f{}_\leftarrow \rightarrow \mathbf{Val}$ gives the values of the input nodes (that are set outside of the fragment);
- $\psi^f : \mathit{Lab}^f{}_* \rightarrow \mathbf{Val}$ gives the values that are set during the initialization.

In an initial configuration, ρ^f and ϕ^f map all nodes to $\perp$ or false (depending on the types of operations in nodes).

Similarly to the dependency graph, all nodes have associated step functions. For the nodes with operations, occurring in the dependency graph (all the operation except for InputB, InputS, OutputB, and OutputS), the step function is the same as defined for the dependency graph.

If $\lambda(v) \in \{\text{InputB}, \text{InputS}\}$ then $f^v((\langle\rho^f, \psi^f, \phi^f\rangle) = \phi^f(v)$. If $\lambda(v) \in \{\text{OutputB}, \text{OutputS}\}$ and u is the source node for the dependency of v, then $f^v(\langle\rho, \psi, \phi\rangle)$ is $\rho(u)$.

The parallel application of all the step functions for nodes gives us a step function of the whole configuration:

$$f(\langle\rho^f, \phi^f, \psi^f\rangle) = \langle\rho^f[v \mapsto f^v(\langle\rho^f, \phi^f, \psi^f\rangle)], \phi^f, \psi^f\rangle \ .$$

The function f is monotone and continuous.

The execution of a dependency graph fragment H, in parallel with the driver algorithm A proceeds as follows:

1. ρ^f is set to map every dependency to $\perp$ / false. ψ is initialized with the (uniformly generated) random coins used in the execution. For each v with $\lambda(v) = \text{RS}$, the value $\psi^f(v)$ is a sufficiently long random bit-string that is independent from every other value. The mapping ϕ^f (containing information on inputs of the fragment) is set to map every InputB operation to false and every InputS operation to $\perp$.

2. A is invoked with the internal state it output during the previous invocation (if this is the first invocation then the internal state is empty) and with the mapping $\rho|_{Lab^f{}_{\rightarrow}}$.

3. If A indicates to stop then stop the execution. Otherwise, A produces a new internal state and a new mapping $\phi' : Lab^f{}_{\leftarrow} \rightarrow \Sigma_\perp$ satisfying $\phi \leq \phi'$. The computational cost of outputting ϕ' is defined to be the number of labels l where $\phi(l) \neq \phi'(l)$.

4. The graph fragment is evaluated — let $\langle\rho', \phi', \psi\rangle$ be the least fixed point of f that is greater or equal to $\langle\rho, \phi', \psi\rangle$. The existence of such fixed point follows from the properties of f.

5. Let $\rho := \rho'$, $\phi := \phi'$. Continue from step 2.

As a result, we get a list of inputs to and outputs from the driver algorithm A. We call the probability distribution (given by the probabilistic generation of ψ and the coin-tosses used by the driver algorithm) over these lists the public view of the driver algorithm A when executed with the given dependency graph fragment H. We denote this distribution by $\mathsf{view}^H(\mathcal{A})$.

6.1.2 Indistinguishability

Two dependency graph fragments, H_1 and H_2 are *indistinguishable* if for all probabilistic polynomial-time driver algorithms $\mathcal{A}$, the distributions $\mathsf{view}^{H_1}(\mathcal{A})$ and $\mathsf{view}^{H_2}(\mathcal{A})$ are indistinguishable. Note that H_1 and H_2 are, in general case, infinite.

6.1.3 Transformations Specification

A graph G contains graph fragment H if:

1. There exists a subgraph G' of G that is isomorphic to H without the input and output nodes;
2. If there is an edge from a node v in G' to some other node in G (including G') that has no corresponding edge in H without the input and output nodes, then there must be an edge from the corresponding node v' in H to some output node.

I.e. the dependency graph fragments indicate the results of which nodes may be used outside that fragment. Also, when matching the fragment onto some graph, we are allowed to use as inputs the outputs of the graph fragment.

The graph transformation is defined by supplying the two fragments. If the graph subject to the transformation contains the first fragment, the graph can be transformed by replacing the first fragment with the second. An example of such a pair of the fragments is given in Figure 11.

6.1.4 Correctness proofs

Theorem 5. If two graph fragments H and H' have computationally indistinguishable public views and graph G contains the fragment H, graph G', constructed from the G by replacing the H with the H', has adversary view, computationally indistinguishable from the G.
Proof. The idea of the proof is to show that from the adversary able to distinguish G from G' it is fairly easy to construct a driver algorithm distinguishing H from H'. But as the latter can only be done with a negligible probability, so the G could be distinguished from G' with the same (negligible) probability.

First, let us define the set of labels used in the below discussion:

- The sets of the labels of the graphs G and G' are categorized into the following sets (on the example of G; the sets for G' are defined in the same way): Lab^G is the set of labels of all the nodes, ${Lab_*}^G$ is the set of labels of all the RS- and Secret-nodes, ${Lab_\leftarrow}^G$ is the set of labels of all the Receive- and req-nodes, ${Lab_\rightarrow}^G$ is the set of labels of all the Send-nodes, and, finally, ${Lab_\bullet}^G$ is $Lab^G \backslash ({Lab_\leftarrow}^G \cup {Lab_*}^G)$.

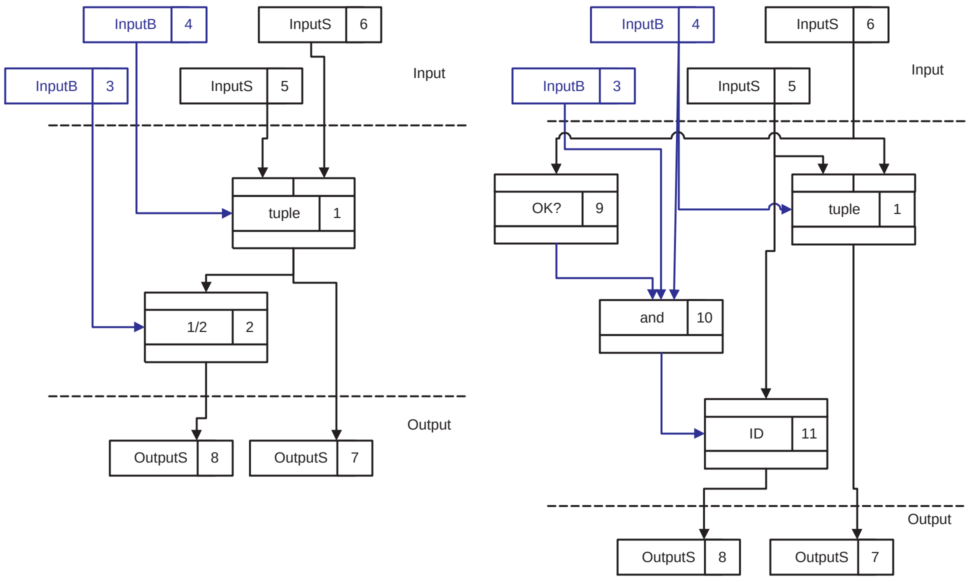

Figure 11: An example of a transformation, graphical form

- The labels of nodes of the graph fragments H and H' are categorized into the following sets (on the example of H; the sets for H' are defined in the same way): Lab^H is the set of labels of all the nodes, ${Lab_*}^H$ is the set of labels of all the RS- and Secret-nodes, ${Lab_\leftarrow}^H$ is the set of labels of all the InputB- and InputS-nodes, ${Lab_\rightarrow}^H$ is the set of labels of all the OutputB- and OutputS-nodes, and, finally, ${Lab_\bullet}^H$ is $Lab^H \backslash ({Lab_\leftarrow}^H \cup {Lab_*}^H)$.

Note that as it is possible to replace H with H', the sets of input and output nodes of H and H' are equal: ${Lab_\leftarrow}^H = {Lab_\leftarrow}^{H'}$ and ${Lab_\rightarrow}^H = {Lab_\rightarrow}^{H'}$.

Suppose there is an adversary $\mathcal{A}_G$, able to distinguish G from G'. Let $S_{\mathcal{A}_G}$ be the state of $\mathcal{A}_G$. During each invocation, $\mathcal{A}_G$ receives $\rho|_{Lab_\rightarrow}$ on input and produces either command stop or the mapping $\phi' : Lab_\leftarrow \rightarrow \Sigma_\perp$ on output. Let $\mathsf{view}^G_M(\mathcal{A}_G)$ and $\mathsf{view}^{G'}_M(\mathcal{A}_G)$ be the adversary views produced by executing the $S_{\mathcal{A}_G}$ in parallel with the G and the G', correspondingly. Suppose that the distribution $\mathsf{view}^G_M(\mathcal{A}_G)$ can be distinguished from the $\mathsf{view}^{G'}_M(\mathcal{A}_G)$ with some probability.

The driver algorithm $\mathcal{A}_H$ for distinguishing the H from the H' is constructed as follows:

1. The state $S_{\mathcal{A}_H}$ consists of the following components:
   (a) $S_{\mathcal{A}_G}$ – The state of the $\mathcal{A}_G$
   (b) $\rho^{\mathcal{A}_H} : {Lab_\bullet}^G \backslash \{Lab^H \backslash \{{Lab_\leftarrow}^H \cup {Lab_\rightarrow}^H\}\} \rightarrow \mathbf{Val}$

   (c) $\phi^{\mathcal{A}_H} : {Lab_{\leftarrow}}^G \rightarrow \mathbf{Val}$
   (d) $\psi^{\mathcal{A}_H} : {Lab_*}^G \backslash {Lab_*}^H \rightarrow \Sigma_\perp$

2. The input to the each algorithm invocation is the state $S_{\mathcal{A}_H}$ and the mapping $\rho^H|_{{Lab_\rightarrow}^H}$

3. The output of the algorithm invocation could be either an indication to stop the execution, or a new internal state and a new mapping $\phi'^H : {Lab_\leftarrow}^H \rightarrow \Sigma_\perp$

4. The procedure performed during the algorithm invocation is the following:

   (a) If the $S_{\mathcal{A}_H}$ is empty (it is the first invocation), initialize the state. The $S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}$ and $S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}$ are set to map every bit string value to $\perp$ and every boolean value to $false$. The $S_{\mathcal{A}_H}.\psi^{\mathcal{A}_H}$ is initialized with the (uniformly generated) random coins used in the execution.
   (b) The portion of the graph G laying outside of H is evaluated — let $\langle {\rho^{\mathcal{A}_H}}', S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}, S_{\mathcal{A}_H}.\psi^{\mathcal{A}_H} \rangle$ be the least fixed point of f (The step-function for defined for all nodes in ${Lab_\bullet}^G \backslash \{Lab^H \backslash \{{Lab_\leftarrow}^H \cup {Lab_\rightarrow}^H\}\}$) that is greater or equal to $\langle S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}, S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}, S_{\mathcal{A}_H}.\psi^{\mathcal{A}_H} \rangle$.
   (c) If ${\rho^{\mathcal{A}_H}}' > S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}$ then let $S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H} = {\rho^{\mathcal{A}_H}}'$; return the $S_{\mathcal{A}_H}$ and $S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}$.
   (d) If ${\rho^{\mathcal{A}_H}}' = S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}$ then
      i. Invoke $\mathcal{A}_G$, passing $S_{\mathcal{A}_H}.S_{\mathcal{A}_G}$ and $S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}|_{{Lab_\rightarrow}^G}$ as parameters
      ii. If $\mathcal{A}_G$ indicates to stop, return the "stop" command.
      iii. If $\mathcal{A}_G$ returns the state $S'_{\mathcal{A}_G}$ and the mapping ${\phi^{\mathcal{A}_G}}' : {Lab_\leftarrow}^G \rightarrow$ **Val**, set $S_{\mathcal{A}_H}.S_{\mathcal{A}_G} = S'_{\mathcal{A}_G}$ and update the $S_{\mathcal{A}_H}.\rho^{\mathcal{A}_H}$ and $S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}$ with the values changed in ${\phi^{\mathcal{A}_H}}'$; return the $S_{\mathcal{A}_H}$ and $S_{\mathcal{A}_H}.\phi^{\mathcal{A}_H}$.

Now let us consider the execution (defined in sec.4.3) of the dependency graph G in parallel with $S_{\mathcal{A}_G}$. Note that some of the steps are split into two. Namely, we separate all the nodes into two groups — the ones belonging to the H (except for input and output nodes), and the rest of the nodes; and first perform the computations related to one group, followed by the computations from the other one. When computing the fixed point, we repeat both steps iteratively until the fixed point is reached. As we are free to define the order of computations, choosing this particular order have no influence on the result of the computation.

1. Initialization

   (a) Initialization of ρ^G is performed in two steps:

      i. ρ^G is set to map every $l \in \mathit{Lab}_{\bullet}{}^G \backslash \{\mathit{Lab}^H \backslash \{\mathit{Lab}_{\leftarrow}{}^H \cup \mathit{Lab}_{\rightarrow}{}^H\}\}$ to $\perp$ / false

      ii. ρ^G is set to map every $l \in \mathit{Lab}^H \backslash \{\mathit{Lab}_{\leftarrow}{}^H \cup \mathit{Lab}_{\rightarrow}{}^H\}$ to $\perp$ / false

   (b) Initialization of ψ^G is also done in two steps:

      i. ψ^G is set to map every $l \in \mathit{Lab}_*{}^G \backslash \mathit{Lab}_*{}^H$ to uniformly generated random coins

      ii. ψ^G is set to map every $l \in \mathit{Labp}^H$ to uniformly generated random coins

   (c) The mapping ϕ^G is set to map every req operation to false and every Receive operation to $\perp$.

2. The $S_{\mathcal{A}_G}$ is invoked with the internal state it output during the previous invocation (if this is the first invocation then the interal state is empty) and with the mapping $\rho^G|_{\mathit{Lab}_{\rightarrow}{}^G}$.

3. If the adversary indicates to stop then stop the execution. Otherwise, the adversary produces a new internal state and a new mapping $\phi^{G'} : \mathit{Lab}_{\leftarrow}{}^G \rightarrow \Sigma_{\perp}$.

4. The graph G is evaluated — let $\langle \rho^{G'}, \phi^{G'}, \psi^G \rangle$ be the least fixed point of f (step-function for the whole graph G) that is greater or equal to $\langle \rho^G, \phi^{G'}, \psi^G \rangle$. The computation of $\rho^{G'}$ can be performed by repeating of the following steps until the fixed point is reached (i.e. once the $\rho^{G^3} = \rho^G$ we set $\rho^{G'} := \rho^G$ and go to the step 5):

   (a) Let f^H be the step-function for the graph fragment H.
   Let $\langle \rho^{G^2}, \phi^{G'}, \psi^G \rangle$ be the least fixed point of f^H that is greater or equal to $\langle \rho^G, \phi^{G'}, \psi^G \rangle$.

   (b) Let f^G be the step-function defined for the nodes belonging to G but not belonging to H (except for input and output nodes). Let $\langle \rho^{G^3}, \phi^{G'}, \psi^G \rangle$ be the least fixed point of f^G that is greater or equal to $\langle \rho^G, \phi^{G^2}, \psi^G \rangle$.

5. Let $\rho^G := \rho^{G'}$, $\phi^G := \phi' G$. Continue from step 2.

The result is the distribution $\mathsf{view}^G_M(\mathcal{A})$ — the probability distribution (given by the probabilistic generation of ψ^G and the adversary's coin-tosses) over the ϕ^G and $\rho^G|^G_{\mathit{Lab}_{\rightarrow}}$.

It is easy to verify that the same computations in the same order are performed when the graph fragment H is executed in parallel with the driver algorithm $\mathcal{A}_H$ — for each step in the execution of G there is a corresponding step in execution of H. Initialization steps 1.(a).ii and 1.(b).ii are performed during the initialization of the H execution. Then, upon the first invocation of $\mathcal{A}_H$, the steps 1.(a).i, 1.(b).i, and 1.(c) are performed (in step 4.(a) of $\mathcal{A}_H$). Step 4.(b) of the first invocation of $\mathcal{A}_H$ does not modify the configuration (as every control dependency is set to $\perp$. The $\mathcal{A}_G$ is invoked in step 2 of the G execution and in the step 4.(d) of the $\mathcal{A}_H$. Evaluation of G, performed in step 4 of the G execution, corresponds to a sequence of steps 4.(c) of the $\mathcal{A}_H$ and the evaluation of H, which are performed until the fixed point is reached. Finally, once the $\mathcal{A}_G$ indicates to stop the execution, it is stopped in both cases.

So, the execution of the H in parallel with $\mathcal{A}_H$ results in the same distribution of $\mathsf{view}_M^G(\mathcal{A})$ of the ϕ^G and $\rho^G|_{Lab_\rightarrow}^G$ components of the driver algorithm state. Therefore, if the $\mathcal{A}_G$ gives two distributions for G and G', the $\mathcal{A}_H$ returns the same two distributions for H and H'. But, by the requirement to H and H', these distributions could not be distinguished with non-negligible probability, which is what we are required to prove. $\square$

The remainder of this section outlines the strategy used to proof the indistinguishability of the public views of the two graph fragments. The following approach is used:

1. For each public output of the first and the second graph fragment, we write down the formula of its evaluation, according to the semantics of the fragment. The formula may take the inputs of the subgraph as a parameter

2. If the formulae for the particular output of the first and the second fragment is the same, the semantics is obviously equal

3. If the formulae differ, then we show that the values computed by them are computationally indistinguishable. While doing so, we usually rely on the properties of the underlying cryptographic primitives

Additionally, as the analyzer performs the transformations over the dependency graph (i.e. given the graph G it produces the graph G'), we show that each of these transformations consists of no more than replacing the graph fragments.

6.2 Dependency Graph Fragment Representation

Like DG, the DGF can be infinite. The DGF structure is regular enough to be represented by a finite graph. We call this finite representation Dependency Graph Fragment Representation, or DGFR.

Formally, the DGFR comprises nodes and edges. Each DGFR node v has:

- $\ell(v)$ — identity;
- $\lambda^{gf}(v)$ — operation;
- $r(v)(D)$ — number of coordinates the node has in each dimension.

Each edge e on the fragment representation has:

- $s(e)$ — edge source;
- $t(e)$ — edge target;
- $port(e)$ — identifier of the input port on the edge target;
- $m(e)(D)$ — function mapping the source to target coordinate indices.

The procedure converting the DGFR to DGF is the same as the procedure converting DGR to DG.

6.2.1 Semantics

The semantics of the DGFR is equal to the semantics of the DGF, represented by it.

6.2.2 Indistinguishability

Two DGFRs are computationally indistinguishable if the DGFs, represented by them, are computationally indistinguishable.

6.2.3 Transformations Specification

The idea of the process of the transformation is following — if the DG, represented by the DGR, contains a sub-graph, isomorphic to DGF, represented by the first DGFR without input and output nodes, then the DGR is replaced with new DGR, representing the DG with the sub-graph replaced with the DGF, represented by the second DGFR.

The sub-graph of DG, represented by the DGR, is isomorphic to the DGF, represented by DGFR, if

- For each DGFR node there is corresponding DGR node;

- For each DGFR edge there is corresponding DGR edge;
- For each DGFR node replication dimension mapping there is a corresponding node dimension mapping on DGR;
- For each DGR edge dimension mapping there is a corresponding edge dimension mapping on DGR.

Additionally, we require that if the DGR contains an edge originating in the node, which corresponds to DGFR node, and the DGFR does not contain the corresponding edge, there should be an edge from the DGFR node to some output node. The transformation is thus specified by a pair of DGFRs. If the DG contains a sub-graph isomorphic to the first DGFR, the DGR of the transformed sub-graph is obtained by replacing the nodes of the sub-graph with the second DGFR, with corresponding dimension mappings.

6.2.4 Correctness proofs

Theorem 6. If two DGFRs fr_1 and fr_2 are computationally indistinguishable and DGR gr contains the fragment fr_1, the DG g', corresponding to DGR gr', constructed from the gr by replacing the fr_1 with the fr_2, has adversary view, computationally indistinguishable from the DG g, corresponding to the DGR gr.
Proof sketch Per definition of the DGFR indistinguishability, the DGFs f_1 and f_2, corresponding to fr_1 and fr_2, have indistinguishable public views. Therefore, replacing the f_1 with f_2 in DG g, corresponding to DGR gr, results in DG g'', having adversary view, computationally indistinguishable from the DG g. Therefore, all we have to prove is that the gr', constructed by DGFR replacement, will also correspond to g'', constructed by DGF replacement. In order to prove that, for each node on g' we should allocate corresponding node on g''.

First, let us consider DGFR transformation: in the gr we find fr'_1, isomorphic to fr_1 without input and output nodes. For each combination of replication dimensions of fr_1 nodes we have corresponding combination of replication dimensions of fr'_1 nodes. During the transformation the nodes belonging to fr'_1 are removed from the gr, and the nodes and edges belonging to fr'_2 (the graph, isomorphic to fr_2, but with replication dimensions changed using the same function which maps the dimensions of fr_1 nodes to fr'_1 nodes) are added.

But exactly the same operation is performed on g — as the f_1 is represented by fr_1 (with fixed dimension mappings), the f_2 — by fr_2 (with the same dimension mapping), and the rest of the g nodes are left untouched.

So, the DGs g' and g'' are isomorphic, therefore, the g'' is represented by gr', and the semantics of the DGs, represented by gr and gr', are computationally indistinguishable. $\square$

7 Transformations

In this section we describe the DGR transformations we used for protocol analysis.

We mention all the transformations we applied, while describe in details only the most representative of them. The Appendix 2 contains the detailed descriptions of all the transformations.

7.1 Dead Code Removal

One of the most trivial transformations is removal of the dead code. Despite being trivial it stands separately of other transformations, as it is the generic (only operation-independent) transformation that makes the DGR smaller (other transformations only add additional operations to the graph.)

A node in a dependency graph is *live* if it is a Send-node or if the value produced by it is consumed by a live node. All nodes that are not live may be removed from the graph.

Formally, let the initial DGFR be the following:

- Inputs
  - I_i, for $1 \leq i \leq n$, the operation is either InputB or InputS and the replication dimensions are given by ι_i;
- Regular nodes
  - A, any operation, except for Send; the replication dimensions are given by α. Let the node have n input ports; the input of the i-th input port is I_i, with coordinate mapping $\lambda_{I_i A}$;
- The DGFR has no outputs.

The DGFR can be replaced with the following one:

- Inputs
  - I_i, for $1 \leq i \leq n$, the operation is either InputB or InputS and the replication dimensions are given by ι_i;
- The DGFR has no regular nodes or outputs.

It is obvious that the adversary view of the resulting graph is the same as for the initial graph (since A is not used to compute anything which is made available to the adversary).

7.2 Local Simplifications

The transformations belonging to this group are based on the properties of the operation semantics. Most of these transformations only require analyzing the arguments of the operation being transformed.

The typical transformation from this group is replacing the sequence of Tuple and Proj operations with the copy of corresponding argument of the Tuple operation. Note that in order for the Proj operation to return non-$\perp$ bit string, just having the corresponding input of the Tuple equal to that bit string is not enough – the other inputs of of the Tuple also need to be different from $\perp$. The control input of the Id operation, introduced during the transformation, reflects this.

Formally, let the initial DGFR be the following:

- Inputs
  - I_i, for $1 \leq i \leq n$, the operation is either InputB or InputS and the replication dimensions are given by ι_i;
  - C_T, InputB operation and replication dimensions ι_{C_T};
  - C_P, InputB operation and replication dimensions ι_{C_P};
- Regular nodes
  - T, Tuple operation; replication dimensions α; the control input is C_T, with coordinate mapping $\lambda_{C_T T}$; let the node have n input ports; the input of the i-th input port is I_i, with coordinate mapping $\lambda_{I_i T}$;
  - P, Proj_i^n operation (for some i, $1 \leq i \leq n$); replication dimensions β; the control input is C_P, with coordinate mapping $\lambda_{C_P P}$; the data input is T, with coordinate mapping λ_{TP};
- Outputs
  - O_T, OutputS operation, replication dimensions α, the node input is T, with identity coordinate mapping;
  - O_P, OutputS operation, replication dimensions β, the node input is P, with identity coordinate mapping.

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - T, same as in the initial DGFR;

  - OK_T, IsOK operation; replication dimensions α; the node input is T, with identity coordinate mapping;
  - A, And operation with two inputs; replication dimensions β; the node inputs are OK_T (coordinate mapping λ_{TP}) and C_P (coordinate mapping $\lambda_{C_P P}$);
  - P', *opid* operation; replication dimensions β; the control input is A, with identity coordinate mapping; the node data input is I_i, with coordinate mapping $\lambda_{I_i T} \circ \lambda_{TP}$;

- Outputs
  - O_T, same as in the initial DGFR; the node input is T, with identity coordinate mapping;
  - O_P, OutputS operation, replication dimensions β, the node input is P', with identity coordinate mapping.

In order to verify that the initial and transformed DGFRs are indistinguishable, let us examine the semantic functions of the public outputs. For O_T the semantics are clearly the same, as the node and all its predecessors are identical in the initial and transformed DGFRs. The semantics of the O_P node are:

- In the initial DGFR, it is equal to the i-th component of the tuple (I_i), if the control input of the projection node (C_P) is equal to true, and the data input of the projection node (T) is n-tuple (which means, it has to be different from $\perp$). If any of the above mentioned conditions does not hold, the O_P is $\perp$;

- In the transformed DGFR, the O_P is equal to the I_i, if and only if the control dependency of the node P' is true, which is only the case if the value of OK_T is true (the result of T is different from $\perp$), and C_P is equal to true. Otherwise, the O_P is $\perp$.

As we see, the O_P semantics are the same in the initial and the transformed DGFRs, therefore the O_P will always have the same value on both of the fragments. Having shown that the public outputs of the fragments are always the same, we have demonstrated that they cannot be distinguished.

7.2.1 Other Bit String Operation Simplifications

The other transformation similar (in terms of the type of the underlying semantics and graph analysis required to perform it) are:

Simplifying Equalities
Two constructor-type nodes (which tag the produced value with its type), having the different operations always produce different bit strings. Two constructor-type nodes, having the same operation produce the same bit string if and only if all their inputs are the same. Two random coins (produced by RS-node) are always not different. We can use these properties in order to replace the IsEq and IsNeq operations either with constants or with a comparisons of the original operations' inputs. Since all the operations, including the IsNeq, are strict ($\mathsf{IsNeq}(\perp, \perp) = \mathsf{false}$), when replacing the IsNeq operation we always check that its inputs are different from $\perp$;

Simplifying the Computations Resulting in Error
The bit string operations performing computations according to some algorithm (a Proj is a good example) check if their inputs are tagged by the corresponding constructors, so it the input is produced by the "wrong" constructor, the operation will always return $\perp$. All the bit string operations have strict semantics — if any of the data inputs is $\perp$, or the control input is false, the result is always $\perp$ (or false, for the bit string operations returning booleans). Using these properties we are able to replace some of the operations with constants (false or $\perp$);

Simplifying the IsOK Operations
The operation can only succeed (return bit string different from $\perp$) if all it arguments are different from false / $\perp$. Therefore, the IsOK of the operation result can be replaced with the conjunction of IsOK of the operation arguments and its control input. In case one of the arguments has invalid type or the control input is false, the IsOK can be replaced with constant (False);

Simplifying the Boolean Algebra Operations
The operations And, Or, and Longor obey the rules of boolean algebra. If one of the And inputs is false, then the operation result is always false (the dual statement holds for the Or operation. The And or Or operation having a single input can be replaced with that input. The sequence of two And operations (a dual statements hold for Or and Longor) can be replaced with a single operation, with a union of the operations inputs. Finally, the sequence of Or and And operations can be modified according to the distribution law ($A \wedge B \vee A \wedge C = A \wedge (B \vee C)$).

7.3 Duplicate Computations Removal

This group consists of a transformations detecting the set of computations always leading to the same result, and, if such a set exists, leaving only one copy of these computation.

The transformations are:

Redundant Coordinate Removal
If a node has a coordinate, which has no corresponding coordinate in each of its inputs (i.e. the values of the node with all other coordinates fixed and this coordinate varying are all equal), this coordinate can be removed from the node (and re-introduced in the coordinate mapping from the node to its successors);

Combining Nodes with Same Inputs
If several nodes have the same set of inputs, all the nodes using the value computed at these nodes can be modified to use the value computed at the first node in the set; As a special case, it makes sense to leave only the one copy of the operation with no inputs (Const, True, False, Error).

7.4 Changing the Computations Order

If some node returns (under some condition) the value of its argument (the examples of such nodes are Id and IfDef), and the result of this node is used in some other computation, the order of these two nodes can be modified to first perform the computation and then take its value.

For example, let the initial DGFR be the following:

- Inputs
  - C_{ID}, InputB operation and replication dimensions $\iota_{C_{ID}}$;
  - I_{ID}, InputS operation and replication dimensions $\iota_{I_{ID}}$;
  - I_i, for $1 \leq i \leq n-1$, the operation is either InputS and the replication dimensions are given by ι_i;
  - C_{OP}, InputB operation and replication dimensions $\iota_{C_{OP}}$;
- Regular nodes
  - ID, Id operation; the replication dimensions are given by α; the control input is C_{ID}, with coordinate mapping $\lambda_{C_{ID}ID}$; the node data input is I_{ID}, with coordinate mapping $\lambda_{I_{ID}ID}$;
  - OP, any operation returning bit string value and having one control and n data inputs; the replication dimensions are given by β; the control input is C_{OP}, with coordinate mapping $\lambda_{C_{OP}OP}$; the input

of the j-th (for a single j, $1 \leq j \leq n$) input port is ID, with coordinate mapping λ_{IDOP}; for all $i \neq j$, $1 \leq i \leq n$ the input of the i-th input port is either I_i with coordinate mapping λ_{I_iOP} (if $i < j$) or I_{i+1} with coordinate mapping $\lambda_{I_{i+1}OP}$ (if $i > j$)

- Outputs
  - O_{ID}, OutputS operation, replication dimension α, the node input is ID, with identity coordinate mapping;
  - O_{OP}, OutputS operation, replication dimension β, the node input is OP, with identity coordinate mapping.

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - ID, Id operation; the replication dimensions are given by α; the control input is C_{ID}, with coordinate mapping $\lambda_{C_{ID}ID}$; the node data input is I_{ID}, with coordinate mapping $\lambda_{I_{ID}ID}$;
  - A, And operation with two inputs; the replication dimensions are given by β; the inputs are C_{OP}, with coordinate mapping $\lambda_{C_{OP}OP}$ and C_{ID}, with coordinate mapping $\lambda_{C_{ID}ID} \circ \lambda_{IDOP}$;
  - OP', same operation as for OP node; the replication dimensions are given by β; the control input is A, with identity coordinate mapping; the input of the j-th input port is I_{ID}, with coordinate mapping $\lambda_{I_{ID}ID} \circ \lambda_{IDOP}$; for all $i \neq j$, $1 \leq i \leq n$ the input of the i-th input port is either I_i with coordinate mapping λ_{I_iOP} (if $i < j$) or I_{i+1} with coordinate mapping $\lambda_{I_{i+1}OP}$ (if $i > j$)
- Outputs
  - O_{ID}, OutputS operation, replication dimension α, the node input is ID, with identity coordinate mapping;
  - $O_{OP'}$, OutputS operation, replication dimension β, the node input is OP, with identity coordinate mapping.

7.5 Cryptographic Primitives

One of the key transformations is making use of the security definition of encryption primitives. Let us consider the asymmetric encryption operation. We require the encryption system to satisfy the IND-CCA2 property as defined in sec. 9.3 — i.e. it should be impossible to distinguish two cipher texts produced by the encryption oracle for two plain texts given by the adversary with non-negligible probability in polynomial time even if the adversary is given the ability to decrypt everything except the challenge cipher text. It means that encryption of the plain text with the given public key could be replaced with the encryption of the string of zeroes (or any other constant) of equal length, and that latter cipher text will be indistinguishable from the first for anyone, except for the one having the corresponding private key. On the decryption side we first check whether the cipher text matches one of the cipher texts already produced, and if it does, the corresponding plain text is returned. If no match is found, the decryption operation is performed.

Formally, let the initial DGFR be the following:

- Inputs
  - C_{RK}, InputB operation, replication dimensions ι_{RK};
  - C_{KP}, InputB operation, replication dimensions ι_{KP};
  - C_{PK}, InputB operation, replication dimensions ι_{PK};
  - C_{RE_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{RE_i}}$;
  - C_{E_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{E_i}}$;
  - C_{D_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions $\iota_{C_{D_j}}$;
  - PT_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{PT_i};
  - CT_j, for $1 \leq j \leq m$, InputS operation, replication dimensions ι_{CT_j};
- Regular nodes
  - RK, RS operation, the replication dimensions α, the control input C_{RK}, with coordinate mapping $\lambda_{C_{RK}RK}$;
  - KP, Keypair operation, the replication dimensions α, the control input C_{KP}, with coordinate mapping $\lambda_{C_{KP}KP}$; the data input RK with identity coordinate mapping;
  - PK, PubKey operation, the replication dimensions α, the control input C_{PK}, with coordinate mapping $\lambda_{C_{PK}PK}$; the data input KP with identity coordinate mapping;
  - RE_i, for $1 \leq i \leq n$, RS operation, the replication dimension β_i, the control input C_{RE_i}, with coordinate mapping $\lambda_{C_{RE_i}RE_i}$;
  - E_i, for $1 \leq i \leq n$, PubEnc operation, the replication dimension β_i, the control input C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$; the data inputs are:
    - Random coins: RE_i, with identity coordinate mapping;
    - Key: PK, with coordinate mapping λ_{PKE_i};

    * Plain text: PT_i, with coordinate mapping $\lambda_{PT_iE_i}$;

  - D_j, for $1 \leq j \leq m$, PubDec operation, the replication dimension γ_j, the control input C_{D_j}, with coordinate mapping $\lambda_{C_{D_j}D_j}$; the data inputs are:

    * Key: KP, with coordinate mapping λ_{PKD_j};
    * Cypher text: CT_j, with coordinate mapping $\lambda_{CT_jD_j}$;

- Outputs

  - O_{PK}, the OutputS operation, the replication dimensions α, the input PK, with identity coordinate mapping;
  - O_{E_i}, for $1 \leq i \leq n$, the OutputS operation, the replication dimensions β_i, the input E_i, with identity coordinate mapping;
  - O_{D_j}, for $1 \leq j \leq m$, the OutputS operation, the replication dimensions γ_j, the input D_j, with identity coordinate mapping;

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;

- Regular nodes

  - RK, RS operation, the replication dimensions α, the control input C_{RK}, with coordinate mapping $\lambda_{C_{RK}RK}$;
  - KP, Keypair operation, the replication dimensions α, the control input C_{KP}, with coordinate mapping $\lambda_{C_{KP}KP}$; the data input RK with identity coordinate mapping;
  - PK, PubKey operation, the replication dimensions α, the control input C_{PK}, with coordinate mapping $\lambda_{C_{PK}PK}$; the data input KP with identity coordinate mapping;
  - RE_i, for $1 \leq i \leq n$, RS operation, the replication dimensions β_i, the control input C_{RE_i}, with coordinate mapping $\lambda_{C_{RE_i}RE_i}$;
  - OK_i, for $1 \leq i \leq n$, IsOK operation, the replication dimensions β_i, the input is PT_i, with coordinate mapping $\lambda_{PT_iE_i}$;
  - A_i, for $1 \leq i \leq n$, And operation, the replication dimensions β_i, the operation has two inputs: C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$, and OK_i, with identity coordinate mapping;
  - E'_i, for $1 \leq i \leq n$, PubEncZ operation, the replication dimensions β_i, the control input A_i, with identity coordinate mapping; the data inputs are:

    * Random coins: RE_i, with identity coordinate mapping;

    - Key: PK, with coordinate mapping λ_{PKE_i};
  - EQ_{ij}, for $1 \le i \le n$ and $1 \le j \le m$, IsEq operation, the replication dimension $\gamma_j + \beta_i$ (pointwise addition), the inputs are:
    - CT_j, with identity coordinate mapping;
    - E'_i, with coordinate mapping $\lambda(d,n).\gamma_j(d)+n$;
  - D_j, for $1 \le j \le m$, PubDec operation, the replication dimension γ_j, the control input C_{D_j}, with coordinate mapping $\lambda_{C_{D_j}D_j}$; the data inputs are:
    - Key: KP, with coordinate mapping λ_{PKD_j};
    - Cypher text: CT_j, with coordinate mapping $\lambda_{CT_jD_j}$;
  - OK_j, for $1 \le j \le m$, IsOK operation, the replication dimension γ_j, the input D_j, with identity coordinate mapping;
  - D'_j, $1 \le j \le m$, IfDef operation, the replication dimension γ_j, the control input C_{D_j}, with coordinate mapping $\lambda_{C_{D_j}D_j}$; the operation has $n+1$ pair of data inputs:
    - for $1 \le i \le n$, the check-input is EQ_{ij}, with the identity coordinate mapping; the data-input is PT_i, with coordinate mapping $\lambda(d,n).\gamma_j(d) + \lambda_{PT_iE_i}$; the contracted dimensions are β_i
    - for $i+1$-th pair, the check-input is OK_j, with identity coordinate mapping; the data-input is D_j, with identity coordinate mapping; there are no contracted dimensions;

- Outputs
  - O_{PK}, the OutputS operation, the replication dimensions α, the input PK, with identity coordinate mapping;
  - O_{E_i}, for $1 \le i \le n$, the OutputS operation, the replication dimensions β, the input E'_i, with identity coordinate mapping;
  - O_{D_j}, for $1 \le j \le m$, the OutputS operation, the replication dimensions γ, the input D'_j, with identity coordinate mapping;

Theorem 7. The public view of the transformed DGF H' (corresponding to the transformed DGFR) is indistinguishable from the public view initial DGF H (corresponding to the initial DGFR).

Proof. In the H' the $\rho[O_{PK}]$ returns the public key, computed in the same way as in the H; $\rho[O_{E_i}]$ returns the encryption of the string of zeroes of the length equal to the length of the plain text $\rho[PT_i]$; the $\rho[O_{D_j}]$ is either one of the plain texts earlier encrypted in the DGF, or the result of the decryption operation (if the cipher text being decrypted has not been encrypted in the DGF).

Also it is possible that $\rho[O_{D_j}]$ would be equal to $\top$. The IfDef operation can return $\top$ only in two cases:

- More than one node EQ_{ij} (comparison operation) returns true; the probability that it happens is negligible, as the values compared with (the result of the node E'_i) are functions of independent random coins (generated at the nodes with labels RE_i).

- Some node EQ_{ij} returns true and the result of the decryption semantics of the PubEnc operation (it will return $\perp$ if the result of the decryption is $\mathbf{0}$).

Having shown that $\rho[O_{D_j}]$ can be equal to $\top$ only with negligible probability, we go on with demonstrating that from the driver algorithm $\mathcal{A}_H$, producing the public views of H from H' and the algorithm A, distinguishing these public views, the algorithm $\mathcal{A}_E$, winning the IND-CCA2 game (as defined in sec. 9.3) with the same probability, can be constructed.

The $\mathcal{A}_E$ behaves like $\mathcal{A}_H$, but instead of returning to the graph fragment evaluation function, it calls the oracle defined in sec. 9.3:

- When $\mathcal{A}_H$ first sets the $\rho[C_{KP}]$, $\rho[C_{RK}]$ to true the $\mathcal{A}_E$ executes $\underline{\text{Initialize}}$ and saves the returned public key,

- when $\mathcal{A}_H$ first sets the $\rho[C_{PK}]$ to true the $\mathcal{A}_E$ sets the $\rho[O_{PK}]$ to the saved public key.

- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv}$) the $\rho[C_{E_i}.\overline{cv}]$, $\rho[C_{RE_i}.\overline{cv}]$ to true and the $\rho[PT_i.\overline{cv}]$ to M_0, the $\mathcal{A}_E$ executes $\underline{\text{LR}}(M_0,\ (0)^{|M_0|})$, and puts the returned value to $\rho[O_{E_i}.\overline{cv}]$,

- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv'}$) the $\rho[C_{D_j}].\overline{cv'}$ to true and the $\rho[CT_j].\overline{cv'}$ to C, the $\mathcal{A}_E$ executes $\underline{\text{Dec}}(C)$, and puts the returned value to $\rho[O_{D_j}].\overline{cv'}$,

- finally, when $\mathcal{A}_H$ indicates to stop with the public view, and the algorithm A is executed to produce the bit b, indicating whether the public view corresponds to H or H', the $\mathcal{A}_E$ calls $\underline{\text{Finalize}}(b)$.

By examining the semantics of the graph fragment step functions, it can be checked that indeed the evaluation of H or H' will set the outputs according to the rules implemented by $\mathcal{A}_E$ above. As it is assumed that all the random coins, including those used by encryption oracle and by the RS operation on the graphs, are generated according to the same distribution, so the key pairs generated in node KP and encryption oracle in the game are.

So, the $\mathcal{A}_E$ will win the IND-CCA2 game with the same probability as the A will distinguish the view $\mathsf{view}^H(\mathcal{A}_H)$ from $\mathsf{view}^{H'}(\mathcal{A}_H)$. By the requirement we put on the encryption scheme, this probability is negligible. $\square$

The similar transformations are defined for symmetric encryption and digital signature schemes.

7.6 Implied Analysis

During this analysis the relations between the results of different operations, following from the semantics of these operations, are made explicit. For instance, the result of an And operation being true implies that each argument of this operation has also to be true.

These relations are documented by introducing a special node with operation $\Rightarrow(A, B)$. This node stands for the sequence of two nodes:

- C, the operation is Before′, the inputs are:
  - A, any operation returning boolean value;
  - B, any operation returning boolean value;
- D, the operation is Halt, the input is C.

The semantics of the Before′ is the following: a node Before′(v_1, v_2) equals false. But:

- If, after a fix-point computation, the value v_1 is true and the value v_2 is false, then the node is replaced with a true-node;
- if, after a fix-point computation, the value v_2 is true, then the node is replaced with a false-node.

If any of the Before′-nodes were replaced with true, then the fix-point computation is repeated.

The semantics of the Halt is defined as:

- Halt$(v) =$ true if $v =$ false
- Halt$(v) = \top$ if $v =$ true

Therefore, the $\Rightarrow(A, B)$ indicates that if A is true, B should also be true, or the DG evaluation will be stopped.

7.6.1 Introducing the $\Rightarrow$ Nodes

We introduce the $\Rightarrow$ node if value of one node being true (in case of boolean node) or different from $\perp$ (in case of a node returning a bit string — then the IsOK of the node result is true) implies the value of another node being true or non-$\perp$.

This process is done iteratively — initially we document the relations between the nodes and / or their inputs, and then propagate the created $\Rightarrow$ nodes (due to the operation transitivity throughout the graph).

During the initial step the following relations are taken into account:

- A node always implies itself;
- Or, Longor-nodes imply its arguments;
- an argument of a And-node implies the node;
- an argument of a node with strict semantics implies its result (all bit string and bit string-to-boolean operations have strict semantics);
- check-input of an IfDef node with the single pair of inputs implies the result of the operation;

The propagation of the $\Rightarrow$ nodes is done according to the following principles (the process is repeated until no more $\Rightarrow$ operations could be created):

- If the node A implies the node B and the node B implies the node C, then node A implies the node C;
- A node implying the input of the And node also implies the operation result;
- If an input to the Or node implies some node, then the node also implies it;
- Similarly to the above three cases, it is possible to introduce new $\Rightarrow$ operation if the initial $\Rightarrow$ operations are connected to the original nodes through the Longor.

Each of the transformations outlined above, can be “reversed” — the resulting DGFR can be replaced with the initial one (having less $\Rightarrow$ operations). Note, however, that it can only be done if all the nodes created during the transformation are present, as in the absence of the rest of the DGFR-result of the transformation, they can turn to $\top$, thus stopping the execution of the DGF, corresponding to the DGFR.

Once introduced, the $\Rightarrow$ operations can be used to simplify the graph.

An example of such transformation is removing the And inputs, dependent on each other.

7.6.2 Removal of the And inputs, depending on each other

If an input of the the And operation implies another input of the same operation, that another input can be removed from the inputs list.

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, the operation is InputB and the replication dimensions are given by ι_i;
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_{B_iA} be the coordinate mapping of i-th input ($1 \leq i \leq n$).
  - D, the operation is $\Rightarrow$, the number of dimensions is β. The inputs are:
    - B_{i_1}, with dimension mapping $\lambda_{B_{i_1}D}$
    - B_{i_2}, with dimension mapping $\lambda_{B_{i_2}D}$.
- Outputs
  - O_A, OutputB operation, replication dimensions α, the node input is A, with identity coordinate mapping;

We require that the dimension mappings present in the initial DGFR guarantee the (constant) correspondence of coordinate of the And inputs — i.e. that irrespective of whether we "trace" the coordinate of B_{i_1} to a coordinate of B_{i_2} through the A node, or through the D node, the same coordinate correspondence relations should be obtained.

If the requirements are met, the initial DGFR can be replaced with the new DGFR:

- Inputs — same as in the initial DGFR
- Regular nodes
  - A', And operation with dimensions α, inputs $B_1, \ldots, B_{i_2-1}, B_{i_2+1}, \ldots, B_n$. For i-th input the dimension mapping is λ_{B_iA}.
  - D, the operation is $\Rightarrow$, the number of dimensions is β. The inputs are:
    - B_{i_1}, with dimension mapping $\lambda_{B_{i_1}D}$

    * B_{i_2}, with dimension mapping $\lambda_{B_{i_2}D}$.

- Outputs
  - O_A, OutputB operation, replication dimensions α, the node input is A', with identity coordinate mapping;

7.6.3 Application to Bit String Nodes

The implied analysis can also be applied for simplifying the nodes operating on bit strings. If a control input of some node implies the result of some IsEq node, involving one of the node data inputs, then this data input will (at the moment of the node evaluation) be always equal to the another input of the IsEq node.

In order to take it into account, we introduce the special node Merge(v_1, v_2), having the following semantics:

- Merge$(v_1, v_2) = \perp$ if $v_1 \neq v_2$
- Merge$(v_1, v_2) = v_1$ if $v_1 = v_2$

Similarly to $\Rightarrow$, the Merge is introduced iteratively. During the initial step we look for all the cases when the node control dependency implies the IsEq operation, involving one of its data inputs. If found, we replace this input with the Merge of the original input and the second IsEq input.

Then, we iteratively combine two Merge operations — if one Merge is used as an input to another, then the inputs of the first node could be used as the inputs of the second.

Once all the possible Merge nodes are introduced, we use them in the local transformations. An example of such usage is during the type check, when the input B of some node A is non-constructor-type node (e.g. Receive), while after the implied analysis the input B is replaced with Merge(B, C). If the C is a constructor-type node, we now can decide whether the type of C (and thus B) is acceptable for A or not.

7.7 Not-And Analysis

During the Not-And (NAND) analysis the relations between the results of different nodes, following from the semantics of these operations, are made explicit. The relationships created in the NAND analysis make it explicit that two values cannot be true at the same time. An example of this situation is that two operations — IsEq and IsNeq — having the same inputs cannot both be true.

These relations are documented by introducing a special node with operation Nand(A, B). This node stands for the sequence of two nodes:

- C, the operation is And, the inputs are:
  - A, any operation returning boolean value;
  - B, any operation returning boolean value;
- D, the operation is Halt, the input is C.

Therefore, the Nand(A, B) indicates that A and B cannot be true at the same time, or the DG evaluation will be stopped.

7.7.1 Introducing the Nand Nodes

This process is done iteratively — initially we document the relations between the nodes, and then propagate the created Nand nodes (due to the operation transitivity throughout the graph).

During the initial step the following relations are taken into account:

- False and Error (through the IsOK) nodes have NAND-relation with any node;
- IsEq and IsNeq nodes with the same inputs are in NAND-relation;
- A comparison of a cypher text-input to the PubDec node with a cypher text produced by PubEncZ node is in NAND relation with the (IsOK of the) result of this PubEncZ node. The dual statement holds for SymDec and SymEncZ nodes.

The propagation of the Nand nodes is done according to the following principles (the process is repeated until no more Nand operations could be created):

- if a node is in NAND-relation with a And node input, then it also is in NAND-relation with the And node result;
- if a node is in NAND-relation with a Or or Longor node, then it also is in NAND-relation with the node inputs;
- if a node is in NAND-relation with a control input or (through the IsOK) with a data input of a node returning bit string, then it also is in NAND-relation with the IsOK of this node;

7.7.2 Application to Boolean Nodes

If a node is in NAND-relation with itself, it cannot be true, and thus can be replaced with False.

More complex cases are also possible — for example, if the result of the And node A, having among this arguments the Or node C, implies the node B, and B is in the NAND-relation with one of the C inputs, then this input can be removed from the C.

7.7.3 Application to Bit String Nodes

If an IsOK of a node is in NAND-relation with itself, then the node cannot have non-$\perp$ value, and thus can be replaced with Error.

Each of the transformations outlined above, can be "reversed" — the resulting DGFR can be replaced with the initial one (having less Nand operations). Note, however, that it can only be done if all the nodes created during the transformation are present, as in the absence of the rest of the DGFR-result of the transformation, they can turn to $\top$, thus stopping the execution of the DGF, corresponding to the DGFR.

7.8 Independence Analysis

The idea of this group of transformations is that two nodes, which performing based on the independent inputs (the transitive reflexive closure of their inputs do not intersect), will produce (with non-negligible probability) a different result, if at least one of them uses randomness as its input.

These transformations rely, in addition to the initial DGFR, to the information, indicating which nodes are can potentially influence the value of each DGFR node.

This information is captured by the following functions:

- $(v', m^{-1}(d,n)) \in \mathit{Inputs}(v)$ if DGFR contains an edge from v' to v with coordinate mapping $m(d,n)$.
- The $\mathit{RTCInputs}(v)$ is a reflexive transitive closure of $\mathit{Inputs}(v)$.
- $\mathit{Independent}(v,v')$ is:
  - false if $\mathit{RTCInputs}(v)$ and $\mathit{RTCInputs}(v')$ intersect (note that as the functions return coordinate mappings, those also need to be equal);
  - false, if for one of $v'' \in \{v,v'\}$ the $\mathit{RTCInputs}(v'')$ contains the node with Receive operation;
  - true otherwise.

Additionally, we use the function $Random(v)$, which indicates whether the node v is computed using the random coins. It is defined as following:

- if $\lambda(v) = \mathsf{RS}$ then $Random(v) = \mathsf{true}$;
- if $\lambda(v)$ is Nonce, Keypair, PubEnc, PubEncZ, SymKey, SymEnc, SymEncZ, SigVer, or Signature, with input from v' used as random coins, then $Random(v) = Random(v')$;
- if $\lambda(v)$ is PubKey or VerKey with input from v' used as key pair, then $Random(v) = Random(v')$;
- otherwise, $Random(v) = \mathsf{false}$

If neither of the nodes has the adversary input as its source (as defined by the *Independent* function), the information captured by the above functions is enough to perform the following two transformations.

7.8.1 Replacing the IsEq Operation not Influenced by Adversary Input

Let the initial DGFR contain the following nodes:

- Inputs
  - X_1, InputS operation, replication dimensions ι_{X_1};
  - X_2, InputS operation, replication dimensions ι_{X_2};
- Regular nodes
  - EQ, IsEq operation, replication dimensions δ, and the following inputs:
    - X_1, with coordinate mapping $\lambda X_1 EQ$;
    - X_2, with coordinate mapping $\lambda X_2 EQ$;
- Outputs
  - O_{EQ}, OutputB operation, replication dimensions δ, the node input is EQ, with identity coordinate mapping.

If the nodes X_1 and X_2 cannot influence each other — or, formally, $Independent(X_1, X_2) = \mathsf{true}$, and $Random(X_1) = \mathsf{true}$, then the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes

    - EQ, False operation, no replication dimensions;

- Outputs
    - O_{EQ}, OutputB operation, replication dimensions δ, the node input is EQ, with identity coordinate mapping.

7.8.2 Replacing the IsNeq Operation not Influenced by Adversary Input

Let the initial DGFR contain the following nodes:

- Inputs
    - X_1, InputS operation, replication dimensions ι_{X_1};
    - X_2, InputS operation, replication dimensions ι_{X_2};
- Regular nodes
    - NEQ, IsNeq operation, replication dimensions δ, and the following inputs:
        * X_1, with coordinate mapping $\lambda X_1 NEQ$;
        * X_2, with coordinate mapping $\lambda X_2 NEQ$;
- Outputs
    - O_{NEQ}, OutputB operation, replication dimensions δ, the node input is NEQ, with identity coordinate mapping.

If the nodes X_1 and X_2 cannot influence each other — or, formally, $Independent(X_1, X_2) = \mathsf{true}$, and $Random(X_1) = \mathsf{true}$, then the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
    - OK_1, IsOK operation, replication dimensions ι_{X_1}, input X_1 (identity coordinate mapping);
    - OK_2, IsOK operation, replication dimensions ι_{X_2}, input X_2 (identity coordinate mapping);
    - NEQ, And operation, replication dimensions δ, and the following inputs:
        * OK_1, with coordinate mapping $\lambda X_1 NEQ$;
        * OK_2, with coordinate mapping $\lambda X_2 NEQ$;

- Outputs
    - O_{NEQ}, OutputB operation, replication dimensions δ, the node input is NEQ, with identity coordinate mapping.

7.8.3 Replacing the IsEq Influenced by Adversary Input

If one of the IsEq inputs depends on the Receive-node, we can apply neither of the transformations described above, as the $\mathit{Independent}(X_1, X_2) = \mathsf{false}$.

First, we introduce the function $\mathit{Independent}'$, to capture this condition

- $\mathit{Independent}'(v, v')$ is:
    - false if $\mathit{RTCInputs}(v)$ and $\mathit{RTCInputs}(v')$ intersect (note that as the functions return coordinate mappings, those also need to be equal);
    - false, if for one of $v'' \in \{v, v'\}$ the $\mathit{RTCInputs}(v'')$ contains the node with Receive operation, and for $v''' \in \{v, v'\}, v''' \neq v''$) the $\mathit{RTCInputs}(v'')$ contains the node with Receive operation;
    - true otherwise.

Then, we analyze the condition under which the other IsEq input may influence the adversary view, and add that condition to the IsEq operation.

We introduce the node $\mathsf{I}(c, r)$ (c is a condition input port, r is a value input port) for documenting whether the node r can influence the DG public outputs. The node $\mathsf{I}(c, r)$ documents the fact that if the public view of the two DGs, differing only in the value of $\psi[r]$, can be distinguished, then the $\rho[c]$ must be equal to true.

Formally, the node $\mathsf{I}(c, r)$ on the DGFR gr means that the following game cannot we won with non-negligible probability — it is given the value of $\psi[r]$ and two DGFRs:

- gr', equal to gr, but with additional node T (Halt operation, input c);
- gr'', equal to gr', but using different (but taken according to the same distribution) random coins in the node r

The task is to determine which of two DGFRs uses the given value of $\psi[r]$ (i.e. to distinguish the gr' from gr''.

As the node $\mathsf{I}(c, r)$, returns no value, and has no effect on the "normal" execution of the DGR / DGFR where it occurs, therefore it does not have the execution semantics defined.

The I nodes are introduced iteratively — initially we document the relations between the nodes, and then propagate the created I nodes.

The initial step is performed as following. Let the initial DGFR be:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - R, RS operation, replication dimensions α. The node control input is C, with the coordinate mapping λ_{CR};
- Outputs
  - O_R, OutputS operation, replication dimensions α, the node input is R, with identity coordinate mapping.

The initial DGFR can be replaced with the the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - R, same as in the initial DGFR;
  - T, True operation, no replication dimensions;
  - OK, IsOK operation, replication dimensions α. The operation input is R, with identity coordinate mapping;
  - A, And operation, replication dimensions α. The operation inputs are T and R, both with identity coordinate mappings;
  - O, Or operation, replication dimensions α. The operation input is A, with identity coordinate mapping;
  - I, operation I, replication dimensions α. The operation condition and value inputs are O and R, correspondingly; both with identity coordinate mappings;
- Outputs — same as in the initial DGFR.

Each iteration is formalized as following. Let the initial DGFR be:

- Inputs
  - A_i, $1 \leq i \leq a$, InputB operation, replication dimensions ι_{A_i};
  - C_i, $1 \leq i \leq c$, InputB operation, replication dimensions ι_{C_i};
  - CR, InputB operation, replication dimensions ι_{CR};
  - CV, InputB operation, replication dimensions ι_{CV};
  - DV_i, $1 \leq i \leq v$, InputS operation, replication dimensions ι_{DV_i};
  - CV_i, $1 \leq i \leq k$, InputB operation, replication dimensions ι_{CV_i};

- DV_{ij}, $1 \leq i \leq k$, $1 \leq j \leq k_i$, InputS operation, replication dimensions $\iota_{DV_{ij}}$;
- CV'_i, $1 \leq i \leq k'$, InputB operation, replication dimensions $\iota_{CV'_i}$;
- $V'_i Check_j$, $1 \leq i \leq k'$, $1 \leq j \leq k'_i$, InputB operation, replication dimensions $\iota_{V'_i CD_j}$;
- $V'_i Data_j$, $1 \leq i \leq k'$, $1 \leq j \leq k'_i$, InputS operation, replication dimensions $\iota_{V'_i CD_j}$;
- $V'_i Check_V$, $1 \leq i \leq k'$, InputB operation, replication dimensions β;
- CV''_i, $1 \leq i \leq k''$, InputB operation, replication dimensions $\iota_{CV''_i}$;
- $CV^{(3)}_i$, $1 \leq i \leq k^{(3)}$, InputB operation, replication dimensions $\iota_{CV^{(3)}_i}$;
- $DV^{(4)}_{ij}$, $1 \leq i \leq k^{(4)}$, $1 \leq j \leq k^{(4)}_i$, InputS operation, replication dimensions $\iota_{DV^{(4)}_{ij}}$;

• Regular nodes

- R, RS operation, replication dimensions α. The node control input is C, with the coordinate mapping λ_{CRR}.
- V, any operation returning bit string value; replication dimensions β. Let the operation have v inputs: the i-th input is DV_i, with coordinate mapping $\lambda DV_i V$;
- OK, IsOK operation, replication dimensions γ_{OK}. The input is V, with coordinate mapping λ_{VOK};
- A, And operation, replication dimensions γ_A. The operation has $c+1$ inputs: OK (coordinate mapping λ_{OKA}) and, for $1 \leq i \leq c$, C_i (coordinate mapping $\lambda_{C_i O}$);
- O, Or operation, replication dimensions γ_O. The operation has $a+1$ inputs: A (coordinate mapping λ_{AO}) and, for $1 \leq i \leq a$, A_i (coordinate mapping $\lambda_{A_i O}$);
- I, operation I, replication dimensions γ_I. The operation condition and value inputs are O (coordinate mapping λ_{OI}) and R (coordinate mapping λ_{RI}), correspondingly;
- V_i, $1 \leq i \leq k$, any operation returning bit string value, replication dimensions β_i. The node control input is CV_i, with coordinate mapping $\lambda_{CV_i V_i}$. Let the node have $k_i + 1$ data inputs: V (coordinate mapping λ_{VV_i}) and, for $1 \leq j \leq k_i$, DV_{ij} (coordinate mapping $\lambda_{DV_{ij} V_i}$);

- V_i', $1 \leq i \leq k'$, IfDef operation, replication dimensions β_i'. The node control input is CV_i, with coordinate mapping $\lambda_{CV_i' V_i'}$. Let the node have $k_i' + 1$ pairs of check and data inputs: $V_i' Check_V$ and V (coordinate mapping $\lambda_{VV_i'}$ and contracted dimensions δ_{i0}), and, for $1 \leq j \leq k_i'$, $V_i' Check_j$ and $V_i' Data_j$, (coordinate mapping $\lambda_{V_i' CD_j V_i'}$ and contracted dimensions δ_{ij});
- V_i'', $1 \leq i \leq k''$, Send operation, replication dimensions β_i''. The node control input is CV_i'', with coordinate mapping $\lambda_{CV_i'' V_i''}$. The data input is V, with coordinate mapping $\lambda_{VV_i''}$;
- $V_i^{(3)}$, $1 \leq i \leq k^{(3)}$, Begin or End operation, replication dimensions $\beta_i^{(3)}$. The node control input is $CV_i^{(3)}$, with coordinate mapping $\lambda_{CV_i^{(3)} V_i^{(3)}}$. The data input is V, with coordinate mapping $\lambda_{VV_i^{(3)}}$;
- $V_i^{(4)}$, $1 \leq i \leq k^{(4)}$, an operation returning boolean value, replication dimensions $\beta_i^{(4)}$. Let the node have $k_i^{(4)} + 1$ data inputs: V (coordinate mapping $\lambda_{VV_i^{(4)}}$) and, for $1 \leq j \leq k_i^{(4)}$, $DV_{ij}^{(4)}$ (coordinate mapping $\lambda_{DV_{ij}^{(4)} V_i^{(4)}}$);

- Outputs
  - O_R, OutputS operation, replication dimensions α, the node input is R, with identity coordinate mapping;
  - O_V, OutputS operation, replication dimensions β, the node input is V, with identity coordinate mapping;
  - O_{OK}, OutputB operation, replication dimensions γ_{OK}, the node input is OK, with identity coordinate mapping;
  - O_A, OutputB operation, replication dimensions γ_A, the node input is A, with identity coordinate mapping;
  - O_O, OutputB operation, replication dimensions γ_O, the node input is O, with identity coordinate mapping;
  - O_{V_i}, $1 \leq i \leq k$, OutputS operation, replication dimensions β_i, the node input is V_i, with identity coordinate mapping;
  - $O_{V_i'}$, $1 \leq i \leq k'$, OutputS operation, replication dimensions β_i', the node input is V_i', with identity coordinate mapping;
  - $O_{V_i^{(4)}}$, $1 \leq i \leq k^{(4)}$, OutputB operation, replication dimensions $\beta_i^{(4)}$, the node input is $V_i^{(4)}$, with identity coordinate mapping.

The initial DGFR can be replaced with the the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - R, V, OK, A, O, V_i $(1 \le i \le k)$, V'_i $(1 \le i \le k')$, V''_i $(1 \le i \le k'')$, $V_i^{(3)}$ $(1 \le i \le k^{(3)})$, $V_i^{(4)}$ $(1 \le i \le k^{(4)})$ — same as in the initial DGFR;
  - $NewOK_i$, $1 \le i \le k$, IsOK operation, replication dimensions β_i. The input is V_i, with identity coordinate mapping;
  - $NewA_i$, $1 \le i \le k$, And operation, replication dimensions γ_{NewA_i}. The operation has $c+1$ inputs: $NewOK_i$ (identity coordinate mapping) and, for $1 \le j \le c$, C_j (coordinate mapping $\lambda_{C_j NewA_i}$);
  - $NewOK'_i$, $1 \le i \le k'$, IsOK operation, replication dimensions β'_i. The input is V'_i, with identity coordinate mapping;
  - $NewA'_i$, $1 \le i \le k'$, And operation, replication dimensions $\gamma_{NewA'_i}$. The operation has $c+2$ inputs: $NewOK'_i$ (identity coordinate mapping), $V'_i Check_V$ (identity coordinate mapping), and, for $1 \le j \le c$, C_j (coordinate mapping $\lambda_{C_j NewA'_i}$);
  - $NewA''_i$, $1 \le i \le k''$, And operation, replication dimensions $\gamma_{NewA''_i}$. The operation has $c+2$ inputs: OK (identity coordinate mapping), CV''_i (identity coordinate mapping), and, for $1 \le j \le c$, C_j (coordinate mapping $\lambda_{C_j NewA''_i}$);
  - $NewO$, Or operation, replication dimensions γ_{NewO}. The operation has either $a + k + k' + k''$ or $a + k + k' + k'' + 1$ inputs, depending on $k^{(4)}$ value:
    - A_i, $1 \le i \le a$ (coordinate mapping $\lambda_{A_i O}$);
    - $NewA_i$, $1 \le i \le k$, (coordinate mapping $\lambda_{NewA_i NewO}$);
    - $NewA'_i$, $1 \le i \le k'$, (coordinate mapping $\lambda_{NewA'_i NewO}$);
    - $NewA''_i$, $1 \le i \le k''$, (coordinate mapping $\lambda_{NewA''_i NewO}$);
    - If $k^{(4)} > 0$ then A (coordinate mapping λ_{ANewO}) (if $k^{(4)} = 0$ this input is absent);
  - $NewI$, I operation, replication dimensions γ_{NewI}. The operation condition and value inputs are $NewO$ (coordinate mapping $\lambda_{NewONewI}$) and R (coordinate mapping λ_{RNewI}), correspondingly.
- Outputs — same as in the initial DGFR.

The newly introduced replication dimensions and coordinate can be chosen freely as long as any two coordinates mapped to the same I coordinate, are mapped to the same $NewI$ coordinate.

Finally, the I node can be used for modifying the IsEq node. Let the initial DGFR contain the following nodes:

- Inputs
  - X_1, InputS operation, replication dimensions ι_{X_1};
  - X_2, InputS operation, replication dimensions ι_{X_2};
  - C, InputB operation, replication dimensions δ;
  - R, InputS operation, replication dimensions ι_R;
- Regular nodes
  - I, I operation, replication dimensions γ, and the following inputs:
    - C, with coordinate mapping λCI;
    - R, with coordinate mapping λRI;
  - EQ, IsEq operation, replication dimensions δ, and the following inputs:
    - X_1, with coordinate mapping $\lambda X_1 EQ$;
    - X_2, with coordinate mapping $\lambda X_2 EQ$;
- Outputs
  - O_{EQ}, OutputB operation, replication dimensions δ, the node input is EQ, with identity coordinate mapping.

If the $\mathit{Independent}'(X_1, X_2) = \mathsf{true}$, $\mathit{Random}(X_1) = \mathsf{true}$, and $R \in \mathit{RTCInputs}(X_1)$, then the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - EQ, same as in the initial DGFR;
  - NewEQ, And operation, replication dimensions δ, and the following inputs:
    - EQ, with identity coordinate mapping;
    - C, with identity coordinate mapping;
- Outputs
  - O_{EQ}, OutputB operation, replication dimensions δ, the node input is NewEQ, with identity coordinate mapping.

8 Experimental Results

The automatic protocol analyzer has been implemented. The analyzer works with the protocols specified in the protocol language. First, the protocol is translated to the DGR. Then, the transformations are applied in iterations, until no more transformations are possible.

The resulting DGR can then be easily interpreted by human. For instance, in relation to the confidentiality property, the interpretation is following:

- If the DGR does not contain the Secret operation, then the protocol satisfies it;
- If the CGR still contains it, then the protocol confidentiality could not be proved with this set of transformations.

The analyzer has been applied to several protocols from the secure protocols open repository (`http://www.lsv.ens-cachan.fr/spore/`):

- Needham-Schroeder public key,
- Lowe's fixed version of Needham-Schroeder Public Key,
- Needham-Schroeder secret key,
- Kao-Chow Authentication,
- TMN.

TMN and Needham-Schroeder public key were not proved to be secure (corresponding graphs still contain secret message even after all the transformations applied); for both protocols there are known attacks.

Graphs corresponding to Lowe's fixed version of Needham-Schroeder Public Key and Kao-Chow Authentication were transformed to a form not containing the secret message, thus indicating that confidentiality property holds.

The Needham-Schroeder secret key is considered secure under the condition that previously exchanged key is not compromised;the Denning-Sacco key freshness attack (if the adversary has obtained the key used in one of the previous sessions) was also successfully detected.

Analysis of the Needham-Schroeder-Lowe public key protocol takes 10 minutes on Pentium M 1.60 GHz machine. The current implementation of the protocol analyzer has some room for optimization (mostly — reusing the intermediate calculations between transformations), so the time of the analysis could be significantly reduced.

9 Conclusions and Future Work

In this thesis we have created a framework suitable for computationally sound analysis of the security properties of cryptographic protocols. It is based on finding out the data and control flows on the protocol representation, and modifying the representation based on the semantics of the individual operations and connections between them. We have shown that the dependency graphs, previously used mainly in the optimizing compilers, can be efficiently used as the protocol representation. We have also shown that it is realistic to automate the analysis performed using our framework.

Let us present the overview of the achieved results and provide some thoughs about the directions, in which this work can be continued.

9.1 Using the Dependency Analysis

The common way of presenting the protocols is a process language, similar to [37]. Unfortunately, the notation used in [37] still leaves significant protocol details assumed and not explicitly stated (for example the equality checks happening in the protocol, possible communication across different sessions of the same protocol, etc.), therefore, we started with sec. 3, defining the process language, providing the abilities for specifying all the details, significant for the analysis. The translation rules from this process language to the dependency graphs are given in sec. 5.

Despite the difference between the semantics of the process language (which is defined, quite naturally, in a structural operational semantics style), and the semantics of the dependency graphs (which can only be defined in denotational style, as the order ot the computations performed in the dependency graph is not fixed), it turned out to be relatively easy to create an algorithm for translation from the process language to the dependency graphs and show that it preserves the aspects of the semantics significant for the security analysis (adversary view). This result is described in sec. 5. Note, however, that the translation in the opposite direction (from the dependency graphs to the process language) is not a trivial, as the dependency graphs may contain the computation patterns, which are not easily serializable. Fortunately, this "reverse" translation is not needed for the security analysis we perform.

The dependency graphs can be tailored for the analysis of the important security properties, such as confidentiality (secrecy) and integrity (non-injective and injective correspondence). The flavor of the dependency graphs, amenable to this analysis, is defined in sec. 4. The main principles of the dependency graph operation remain close to those defined in [33], while the semantics, including the concept of the dependency graph *execution* by the adversary, is new. The concept of transforming the adversary games (in our case, these games are defined by the rules according to which the adversary executes the dependency graph (this part is constant and not transformed), and the dependency graph itself (which is amenable to transformation)) has the important advantage over our earlier works ([10, 12]), that the single dependency graph is capable of holding all the information flows between all the runs of the protocols (in multiple sessions). Therefore, each transformation results in exactly *one* game, covering all the cases, and not in a set of games (one game per case).

The developed framework does not dictate the particular set of transformations to be defined and applied during the analysis, as it operates purely on the level of dependency graph fragments (without taking their semantics into the account). The definition of each transformation (two dependency graph fragments, having the equivalent semantics) is the part where the semantics equivalence should be controlled. The set of transformations sufficient for achieving the practical results, described in the previous section, is defined in sec. 7 and Appendix 2.

9.2 Contributions of the Dissertation

The contributions of this dissertation can be summarized as follows:

- A definition of the process language capturing all the details necessary for the analysis of confidentiality and integrity of the protocols;
- A new approach (extensible framework) for the computationally sound analysis of the security protocols, based on a combination of two techniques (dependency graphs and game transformations), previously never used together for this purpose;
- An algorithm, translating the protocol from the process language to a dependency graph;
- A set of dependency graph transformations (input to the framework) sufficient for analysing several well-known protocols:
  - Needham-Schroeder public key,
  - Lowe's fixed version of Needham-Schroeder Public Key,
  - Needham-Schroeder secret key,
  - Kao-Chow Authentication,
  - TMN.

9.3 Future Work

The possible directions for the further research directions are: application of the framework for determining other security properties, enriching the programming language with more cryptographic primitives, calculation of the exact (negligible) probability of distinguishing the semantics of the resulting graph from the initial one, and running the framework on more protocols. In particular, it would be interesting to verify protocols that are not so easily expressible in the Dolev-Yao model because of the cryptographic primitives they are using (e.g. verifiable secret sharing) and the security properties we are trying to prove. Voting schemes (for example, [23]) are obvious candidates for testing our framework.

References

[1] Robert A. Ballance, Arthur B. Maccabe, Karl J. Ottenstein. The Program Dependence Web: A Representation Supporting Control-, Data-, and Demand-Driven Interpretation of Imperative Languages. ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, pages 257–271, 1990.

[2] Jeanne Ferrante, Karl J. Ottenstein, Joe D. Warren. The Program Dependence Graph and Its Use in Optimization. ACM Transactions on Programming Languages and Systems **9**(3):319–349, July 1987.

[3] Keshav Pingali, Micah Beck, Richard Johnson, Mayan Moudgill, Paul Stodghill. Dependence Flow Graphs: an Algebraic Approach to Program Dependencies. Advances in Languages and Compilers for Parallel Processing, , MIT Press, Cambridge, MA, pages 445–467, 1991.

[4] Michael Backes, Birgit Pfitzmann, Michael Waidner. A composable cryptographic library with nested operations. 10th ACM Conference on Computer and Communications Security, pages 220–230, 2003.

[5] Gavin Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. TACAS 1996, LNCS 1055, pages 147–166, 1996.

[6] Roger M. Needham, Michael D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Communications of the ACM **21**(12):993-999, December 1978.

[7] Mihir Bellare, Phillip Rogaway. Entity Authentication and Key Distribution. CRYPTO 1993, LNCS 773, pages 232–249, 1993.

[8] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. CRYPTO '98, LNCS 1462, pages 26–45, 1998.

[9] Mihir Bellare, Chanathip Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ISBN:3-540-41404-5, pages 531–545, 2000.

[10] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. 2004 IEEE Symposium on Security and Privacy, pages 71–85, May 2004.

[11] Ilja Tšahhirov, Jevgeni Bolotov, Jaak Tepandi. Digital signature systems — an interoperability outlook. Scientific proceedings of Riga Technical University, computer science, pages 60–68, 2004.

[12] Ilja Tšahhirov and Peeter Laud. Digital signature in automatic analyses for confidentiality against active adversaries. Nordsec 2005, Proceedings of the 10th Nordic Workshop on Secure IT Systems, pages 29-41, October 2005

[13] Martín Abadi and Phillip Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). International Conference IFIP TCS 2000, LNCS 1872, pages 3–22, August 2000.

[14] Martín Abadi and Jan Jürjens. Formal Eavesdropping and its Computational Interpretation. Theoretical Aspects of Computer Software, 4th International Symposium (TACS 2001), LNCS 2215, pages 82–94, September 2001.

[15] Pedro Adão, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of formal encryption in the presence of key-cycles. 10-th European Symposium on Research in Computer Security (ESORICS 2005), LNCS 3679, pages 374-396, September 2005.

[16] M. Backes and P. Laud. Computationally Sound Secrecy Proofs by Mechanized Flow Analysis. 13th ACM Conference on Computer and Communications Security (CCS 2006), pages 370–379, 2006.

[17] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. Eurocrypt 2006, LNCS 4004, pages 409–426, 2006. (also: Cryptology ePrint Archive, Report 2004/331, `http://eprint.iacr.org`)

[18] B. Blanchet. A computationally sound mechanized prover for security protocols. Proc. 27th IEEE Symposium on Security & Privacy, 2006.

[19] B. Blanchet and D. Pointcheval. Automated Security Proofs with Sequences of Games. CRYPTO 2006, LNCS 4117, pages 537–554, 2006.

[20] B. Blanchet. Computationally Sound Mechanized Proofs of Correspondence Assertions. Proc. 20th IEEE Computer Security Foundations Symposium, 2007.

[21] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001), pages 136–145, October 2001.

[22] Veronique Cortier and Bogdan Warinschi. Computationally Sound, Automated Proofs for Security Protocols. 14th European Symposium on Programming (ESOP 2005), LNCS 3444, pages 157–171, April 2005.

[23] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A Secure and Optimally Efficient Multi-Authority Election Scheme. EUROCRYPT'97, LNCS 1233, pages 103–118, 1997.

[24] Danny Dolev and Andrew C. Yao. On the security of public key protocols. IEEE Transactions on Information Theory, IT-29(12), pages 198–208, March 1983.

[25] Joshua D. Guttman, F. Javier Thayer, and Lenore D. Zuck. The Faithfulness of Abstract Protocol Analysis: Message Authentication. 8th ACM Conference on Computer and Communications Security (CCS 2001), pages 186–195, November 2001.

[26] Peeter Laud. Handling Encryption in Analysis for Secure Information Flow. 12th European Symposium on Programming (ESOP 2003), LNCS 2618, pages 159–173, April 2003.

[27] Peeter Laud. Secrecy Types for a Simulatable Cryptographic Library. 12th ACM Conference on Computer and Communications Security (CCS 2005), pages 26–35, 2005.

[28] Gavin Lowe. A Hierarchy of Authentication Specification. 10th Computer Security Foundations Workshop, pages 31–44, 1997.

[29] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the Picture: Soundness of Formal Encryption in the Presence of Active Adversaries. 14th European Symposium on Programming (ESOP 2005), LNCS 3444, pages 172–185, April 2005.

[30] Daniele Micciancio and Saurabh Panjwani. Adaptive Security of Symbolic Encryption. 2nd Theory of Cryptography Conference (TCC 2005), LNCS 3378, pages 169–187, February 2005.

[31] Daniele Micciancio and Bogdan Warinschi. Completeness Theorems for the Abadi-Rogaway Logic of Encrypted Expressions. Workshop in Issues in the Theory of Security (WITS 2002), January 2002.

[32] B. Pfitzmann, M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. 2001 IEEE Symposium on Security and Privacy (IEEE S&P 2001), pages 184–200, 2001.

[33] K. Pingali and M. Beck and R. Johnson and M. Moudgill and P. Stodghill. Dependence Flow Graphs: an Algebraic Approach to Program Dependencies. Advances in Languages and Compilers for Parallel Processing, MIT Press, Cambridge, MA, pages 445–467, 1991.

[34] Ilja Tšahhirov and Peeter Laud. Application of Dependency Graphs to Security Protocol Analysis. Trustworthy Global Computing (TGC 2007) post-proceedings, Heidelberg: Springer, Berlin, pages 294–311.

[35] Andrew C. Yao. Theory and Applications of Trapdoor Functions (extended abstract). 23rd Annual Symposium on Foundations of Computer Science, pages 80–91, November 1982.

[36] US Code Title 44 § 3542 (b)(1) (2006)

[37] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, february 1989.

Publications by the author

Tšahhirov, I. , Bolotov, J. , Tepandi, J. (2004). Digital signature systems — an interoperability outlook. In *Scientific proceedings of Riga Technical University, computer science, Riga, 2004* (pp.60-190).

Tšahhirov, I. , Laud, P. (2005). Digital signature in automatic analyses for confidentiality against active adversaries. In *Nordsec 2005, Proceedings of the 10th Nordic Workshop on Secure IT Systems, October 20-21, 2005* (pp. 29-41).

Tšahhirov, I. , Laud, P. (2008). Application of Dependency Graphs to Security Protocol Analysis. In *Post-proceeding of Trustworthy Global Computing (TGC 2007), 2008* (pp. 294-311)

Turvaprotokollide analüüs arvutuslikul mudelil — sõltuvusgraafidel põhinev lähenemisviis

Lühikokkuvõte

Käesolevas töös pakutakse välja raamistik turvaprotokollide konfidentsiaalsuse ja tervikluse omaduste analüüsiks. Töös kasutatud konfidentsiaalsuse definitsioon põhineb protokolli avalike väljundite arvutuslikul sõltumatusel salajastest sisenditest. Kuna krüptograafiliste primitiivide omaduste definitsioonid kasutavad tavaliselt arvutuslikku mudelit, on selline lähenemisviis eelistatav informatsiooniteooriale (Dolev-Yao mudelil) tugineval lähenemisel.

Pakutud raamistik baseerub sõltuvusgraafidel ja ründaja mängude teisendamisel, mida ei ole seni turvalisuse analüüsil koos kasutatud. Me lähtume sõltuvusgraafist, kus on esitatud andme- ja kontrollivoo sõltuvused kõikide protokollis toimuvate arvutuste vahel ja asendame krüptograafilisi primitiive konstruktsioonidega, mis on "ilmselt turvalised". Transformatsioone rakendatakse selliselt, et esialgse ja transformeeritud sõltuvusgraafi semantikad oleksid arvutuslikult eristamatud. Transformeeritud graafi analüüsitakse uuesti. Protsess jätkub, kuni jõuame graafini, mida ei ole enam võimalik transformeerida. Protokoll on turvaline, kui selle transformeeritud versioon on turvaline.

Loodud raamistiku praegune versioon sobib nende protokollide analüüsiks, mis kasutavad sümmeetrilist või asümmeetrilise krüpteerimist ja digitaalallkirja. Raamistikku saab laiendada ulatuslikumale operatsioonide ja krüptograafiliste primitiivide hulgale, et analüüsida protokolle, mis neid primitiive kasutavad.

Pakutud raamistik sobib protokollide turvalisuse automatiseeritud tõestamiseks. Raamistiku põhjal on loodud turvaprotokollide automatiseeritud analüsaator. Analüsaatorit on testitud järgmistel protokollidel: Needham-Schroeder public key, Needham-Schroeder-Lowe public key, Needham-Schroeder secret key, Kao-Chow authentication, ja TMN. Testimine näitas, et analüsaator töötab õigesti ning seda võib edaspidi kasutada uute protokollide analüüsiks.

Võtmesõnad: staatiline analüüs, turvaprotokollide analüüs, sõltuvusgraafid.

Security Protocols Analysis in the Computational Model — Dependency Flow Graphs-Based Approach

Abstract

This thesis presents a framework for a static protocol analysis for the confidentiality and integrity properties. The definition of the confidentiality is based on computational independence of the protocol public outputs from its secret inputs, thus allowing to gracefully handle the cryptographic primitives, usually having their security defined in computational, not in the information-theoretical, model.

The framework is a combination of two techniques, which had not been applied together for the protocol analysis before: the dependency flow graphs (widely used in program optimisations) and the game transformations.

We start with the protocol representation as a dependency graph indicating the control and data flows in all possible runs of the protocol and replace the cryptographic operations with constructions, which are "obviously secure". Transformations are made in such a way that the semantics of the resulting dependency graph remains computationally indistinguishable from the initial graph; the transformed graph is analysed again; the transformations are be applied until no more transformations are possible. A protocol is deemed secure if its transformed version is secure.

The technique is presented as an extensible framework, with support for most common operations and several cryptographic primitives (symmetric and asymmetric encryption, digital signatures) already implemented. The framework is extensible with additional operations or cryptographic primitives, in order to analyse the protocols using them.

The framework is well-suited for producing fully-automated proofs for protocol security. The automatic protocol analyser has been implemented and tested on several protocols: Needham-Schroeder public key, Needham-Schroeder-Lowe public key, Needham-Schroeder secret key, Kao-Chow authentication, and TMN. The testing results comply to the publicly available information, so the analyser can be used for new protocols analysis.

Keywords: static analysis, security protocol analysis, dependency graphs.

Curriculum Vitae (in Estonian)

1. Isikuandmed

   Ees- ja perekonnanimi: Ilja Tšahhirov
   Sünniaeg ja -koht: 02.11.1977, Tallinn
   Kodakondsus: Eesti

2. Kontaktandmed

   Aadress: 15 Raja Street, Tallinn 12618
   Telefon: +372 529 2264
   E-post aadress: ilja.tshahhirov@ttu.ee

3. Hariduskäik

Õppeasutus (nimetus lõpetamise ajal)	Lõpetamise aeg	Haridus (eriala/kraad)
Tallinna Tehnikaülikool	2002	M.Sc. (informaatika)
Tallinna Tehnikaülikool	2000	B.Sc. (informaatika)

4. Keelteoskus (alg-, kesk- või kõrgtase)

Keel	Tase
Eesti keel	Kõrgtase
Inglise keel	Kõrgtase
Vene keel	Kõrgtase

5. Teenistuskäik

Töötamise aeg	Tööandja nimetus	Ametikoht
2007-...	Invision Software	Tarkvara arendusjuht
2006-2007	Hansapank	Süsteemide halduse osakonna juhataja
2005-2006	Hansapank	Projektijuht
2005-2006	Tallinna Tehnikaülikool	Erakorraline teadur
2002-2005	Datalane	Projektijuht
1999-2001	Previo Estonia	Vanem tarkvara arendaja
1998-1999	Stac Estonia	Tarkvara arendaja
1997-1998	Datlin Software	Tarkvara kaasarendaja

6. Teadustegevus

   Tšahhirov, I. , Bolotov, J. , Tepandi, J. (2004). Digital signature systems — an interoperability outlook. In *Scientific proceedings of Riga Technical University, computer science, Riga, 2004* (pp.60-190).

Tšahhirov, I. , Laud, P. (2005). Digital signature in automatic analyses for confidentiality against active adversaries. In *Nordsec 2005, Proceedings of the 10th Nordic Workshop on Secure IT Systems, October 20-21, 2005* (pp. 29-41).

Tšahhirov, I. , Laud, P. (2008). Application of Dependency Graphs to Security Protocol Analysis. In *Post-proceeding of Trustworthy Global Computing (TGC 2007), 2008* (pp. 294-311)

7. Kaitstud lõputööd

   "Tooteliini spetsialiseeritud arendus- ja hooldusprotsess"
   M.Sc. (Informaatika), Tallinna Tehnikaülikool, 2002

   "Hajutatud süsteemitehnoloogia. Platvormist sõltumatu lähenemisviis"
   B.Sc. (Informaatika), Tallinna Tehnikaülikool, 2000

8. Teadustöö põhisuunad

   Krüptograafilised meetodid, staatiline analüüs, protokollide analüüs, hajutatud arvutus, süsteemide koostalitlusvõime

Curriculum Vitae

1. Personal data

   Name: Ilja Tšahhirov
   Date and place of birth: 2 November 1977, Tallinn
   Citizenship: Estonian

2. Contact Information

   Address: 15 Raja Street, Tallinn 12618
   Phone: +372 529 2264
   E-mail: ilja.tshahhirov@ttu.ee

3. Education

Educational institution	Graduation year	Education (field of study/degree)
Tallinn University of Technology	2002	M.Sc. (Informatics)
Tallinn University of Technology	2000	B.Sc. (Informatics)

4. Language skills (basic, intermediate or high level)

Language	Level
Estonian	High Level
English	High Level
Russian	High Level

5. Professional Employment

Period	Organisation	Position
2007-...	Invision Software	Software Development Manager
2006-2007	Hansapank	Head of System Administration
2005-2006	Hansapank	Project Manager
2005-2006	Tallinn University of Technology	Extraordinary Researcher
2002-2005	Datalane	Project Manager
1999-2001	Previo Estonia	Senior Software Developer
1998-1999	Stac Estonia	Software Developer
1997-1998	Datlin Software	Associated Software Developer

6. Scientific work

   Tšahhirov, I. , Bolotov, J. , Tepandi, J. (2004). Digital signature systems — an interoperability outlook. In *Scientific proceedings of Riga Technical University, computer science, Riga, 2004* (pp.60-190).

   Tšahhirov, I. , Laud, P. (2005). Digital signature in automatic analyses for confidentiality against active adversaries. In *Nordsec 2005, Proceedings of the 10th Nordic Workshop on Secure IT Systems, October 20-21, 2005* (pp. 29-41).

   Tšahhirov, I. , Laud, P. (2008). Application of Dependency Graphs to Security Protocol Analysis. In *Post-proceeding of Trustworthy Global Computing (TGC 2007), 2008* (pp. 294-311)

7. Defended theses

   "Tooteliini spetsialiseeritud arendus- ja hooldusprotsess"
   ("Specialized process for incremental development and maintenance of a product line")
   M.Sc. (Informatics), Tallinn University of Technology, 2002

   "Hajutatud süsteemitehnoloogia. Platvormist sõltumatu lähenemisviis"
   ("System distribution technology. Inter-platform approach")
   B.Sc. (Informatics), Tallinn University of Technology, 2000

8. Research Interests

   cryptographic methods, static analysis, protocol analysis, distributed computations, system interoperability

Appendix 1: Cryptographic Notions

We require the cryptographic primitives, used in the protocols analyzed, to satisfy certain security requirements. This section contains the formal definition of the cryptographic primitives, which may occur in the protocols analyzed, along with the definition of the requirements put on them. Some of the transformations (defined in sec. 7) refer to these requirements.

Note that the definitions the properties specified in this section are just one possible option. If another set of the cryptographic primitives and/or properties could be considered by the same core framework, the the definitions (and transformations relying on them), need to be adjusted accordingly.

Public Key Encryption

A *asymmetric key encryption scheme* (throughout this paper we also use the term public key encryption scheme), as defined in [8], is given by a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where

- $\mathcal{K}^r(1^k)$, the *key generation algorihtm*, is a probabilistic algorithm that takes a security parameter $k \in \mathbf{N}$ (provided in unary) and returns a pair (pk, sk) of matching public and secret keys. The random coins generated and used by $\mathcal{K}$ are denoted by r.
- $\mathcal{E}^r$, the *encryption algorithm*, is a probabilistic algorithm that takes a public key pk and a message $x \in \{0,1\}^*$ to produce a cipher text y. The random coins generated and used by $\mathcal{E}$ are denoted by r.
- $\mathcal{D}$, the *decryption algorithm*, is a deterministic algorithm which takes a secret key sk and cipher text y to produce either a message $x \in \{0,1\}^*$ or a special symbol $\perp$ to indicate that the cipher text was invalid.

It is also required that for all (pk, sk) which can be output by $\mathcal{K}^r(1^k)$, for all $x \in \{0,1\}^*$, and for all y that can be output by $\mathcal{E}^r_{pk}(x)$, that $\mathcal{D}_{sk}(y) = x$. The $\mathcal{K}$, $\mathcal{E}$, and $\mathcal{D}$ can be computed in polynomial time.

The encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is said to be IND-CCA2-secure, if for every adversary A, which operates in polynomial time, the probability of winning the game, defined below (by guessing the bit b), is a negligible function of a security parameter. The game is defined as an interface to the oracle, encapsulating the asymmetric encryption scheme:

- <u>proc Initialize</u>
  $(sk, pk) \overset{\$}{\leftarrow} \mathcal{K}^r$; $b \overset{\$}{\leftarrow} \{0,1\}$; $S \leftarrow \emptyset$; Return pk
- <u>proc LR(M_0,M_1)</u>
  $C \overset{\$}{\leftarrow} \mathcal{E}^r_{pk}(M_b)$; $S \leftarrow S \cup (C, M_0)$; Return C

- proc Dec(C)
  ```
  If $(C, X) \in S$ then $M \leftarrow X$ else $M \leftarrow \mathcal{D}_{sk}(C)$
  Return $M$
  ```

- proc Finalize(d)
  ```
  Return $(d = b)$
  ```

Note that our definition of IND-CCA2 is slightly different from the one in [8]: We allow calling Dec with the cipher texts previously encrypted with LR, but always return M_0, passed to the corresponding invocation of LR. As M_0 is always returned irrespectively of what b is, this modification does not give any additional information to the adversary.

Secret Key Encryption

A *symmetric key encryption scheme*, per definition in [9], is a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

- $\mathcal{K}^r(1^k)$, the *key generation algorihtm*, is a probabilistic algorithm that takes a security parameter $k \in \mathbf{N}$ and returns the key K. The random coins generated and used by $\mathcal{K}$ are denoted by r.

- $\mathcal{E}^r$, the *encryption algorithm*, is a probabilistic algorithm that takes a key K and a message $x \in \{0,1\}^*$ to produce a cipher text y. The random coins generated and used by $\mathcal{E}$ are denoted by r.

- $\mathcal{D}$, the *decryption algorithm*, is a deterministic algorithm which takes a key K and cipher text y to produce either a message $x \in \{0,1\}^*$ or a special symbol $\perp$ to indicate that the cipher text was invalid.

It is required that for all K which can be output by $\mathcal{K}^r(1^k)$, for all $x \in \{0,1\}^*$, and for all y that can be output by $\mathcal{E}^r_K(x)$, that $\mathcal{D}_K(y) = x$. The $\mathcal{K}$, $\mathcal{E}$, and $\mathcal{D}$ can be computed in polynomial time.

The encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is said to be IND-CCA-secure, if for every adversary A, which operates in polynomial time, the probability of winning the game, defined below (by guessing the bit b), is a negligible function of a security parameter. The game is defined as an interface to the oracle, encapsulating the symmetric encryption scheme:

- proc Initialize
  ```
  $K \xleftarrow{\$} \mathcal{K}^r$; $b \xleftarrow{\$} \{0,1\}$; $S \leftarrow \emptyset$
  ```

- proc LR(M_0,M_1)
  ```
  $C \xleftarrow{\$} \mathcal{E}^r_K(M_b)$; $S \leftarrow S \cup (C, M_0)$; Return $C$
  ```

- proc Dec(C)
  ```
  If $(C, X) \in S$ then $M \leftarrow X$ else $M \leftarrow \mathcal{D}_K(C)$
  Return $M$
  ```

- proc Finalize(d)
  ```
  Return $(d = b)$
  ```

As with the asymmetric encryption scheme, we use the definition of IND-CCA, which is slightly different from [9]: We allow calling Dec with the cipher texts previously encrypted with LR, but always return M_0, passed to the corresponding invocation of LR. As M_0 is always returned irrespectively of what b is, this modification does not give any additional information to the adversary.

Additionally, we require that the symmetric encryption scheme satisfies the *integrity* property, meaning that a valid cipher text cannot be produced with non-negligible probability without knowing the key. The formal definition of this property can be found in [10].

Digital Signature

We use the same definition of the signature scheme as in [12]. A *signature scheme* is a triple of polynomial-time algorithms $(\mathcal{K}, \mathcal{S}, \mathcal{T})$. $\mathcal{K}$ and $\mathcal{S}$ are probabilistic, $\mathcal{T}$ is deterministic. Here $\mathcal{K}$ is the key pair generation algorithm, $\mathcal{S}$ is the signature generation algorithm, and $\mathcal{T}$ is the signature verification algorithm. All algorithms take the security parameter n (represented in unary 1^n) as an argument. Additionally, $\mathcal{S}$ takes two more arguments — secret signing key and message to sign. Also, $\mathcal{T}$ takes three more arguments — public test key, signature, and message to verify this signature with. Let $msg \in \Sigma$. For all pairs (sk, pk) that may be returned by $\mathcal{K}$ and for all signatures s that may be returned by $\mathcal{S}(1^n, sk, msg)$, the algorithm $\mathcal{T}(1^n, pk, s, msg)$ must return true. This definition only states that $\mathcal{T}$ should recognize signatures, correctly generated with $\mathcal{S}$. We also want the signature system to be secure, i.e. forging the signature without possessing the secret key should be hard.

The signature scheme $(\mathcal{K}, \mathcal{S}, \mathcal{T})$ is said to be existentially unforgeable under adaptive chosen message attack (ACMA), if for every adversary A, which operates in polynomial time, the probability of winning the game, defined below, is a negligible function of a security parameter. The game is defined as an interface to the oracle, encapsulating the signature scheme:

- proc Initialize
  ```
  $(sk, pk) \xleftarrow{\$} \mathcal{K}^r(1^n)$; Return $pk$
  ```

- proc Sign(M)
  ```
  $S \xleftarrow{\$} \mathcal{S}^r(1^n, sk, M)$; Return $S$
  ```

-
```
proc Verify($S$, $M$)
$b \leftarrow \mathcal{T}(1^n, pk, S, M)$
Return $b$
```

The goal of the adversary is to finally output two values m and s (meant as a forged signature for the message m) with Verify(s,m)=*true* and where m is not among the messages previously signed by Sign.

Appendix 2: Transformations

This section contains the full list of transformations we used in out analysis. The transformations are specified in the same formal language, and are categorized as in the thesis main part.

Bit String-to-Boolean Nodes Simplifications

Equality of a Node with Itself

Let the initial DGFR be the following:

- Inputs
  - S, InputB operation; replication dimensions ι_S;
- Regular nodes
  - Eq, IsEq operation, with replication dimensions α, and both inputs equal to S, with coordinate mapping λ_{SEq};
- Outputs
  - O_{Eq}, $\mathsf{OutputS}$ operation, with replication dimensions α and input Eq, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - OK, IsOK operation, with replication dimensions ι_S, and input S, with identity coordinate mapping;
- Outputs
  - O_{Eq}, $\mathsf{OutputS}$ operation, with replication dimensions α and input OK, with coordinate mapping λ_{SEq}.

Equality of Values Produced by Different Constructor-Type Operations

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;

  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2j}, for $1 \leq j \leq m$ (m may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2j}}$;

- Regular nodes
  - S_1, any constructor-type operation having n data inputs and returning bit string, replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1} S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i} S_1}$;
  - S_2, any constructor-type operation, which is either different from S_1, or, if S_1 is RS, the S_2 may also be RS. Let the operations have m data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2} S_2}$. The operation j-th data input is S_{2j}, with coordinate mapping $\lambda_{S_{2j} S_2}$;
  - Eq, IsEq operation, the replication dimensions γ. The operation inputs are S_1 (with coordinate mapping $\lambda_{S_1 Eq}$) and S_2 (with coordinate mapping $\lambda_{S_2 Eq}$);

- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;
  - O_{Eq}, OutputB operation, replication dimensions γ, and input Eq, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - S_1, S_2, defined in the same way as on the initial DGFR;
  - F, False operation with no replication;
- Outputs
  - O_{S_1}, O_{S_2}, defined in the same way as on the original DGFR;
  - O_{Eq}, OutputB operation, replication dimensions γ, and input F, with identity coordinate mapping.

Equality of Values Produced by Same Constructor-Type Operation

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2i}}$;
- Regular nodes
  - S_1, any constructor-type operation having n data inputs and returning bit string, replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1}S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}S_1}$;
  - S_2, same operation as for S_1. Let the operations have n data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2}S_2}$. The operation i-th data input is S_{2i}, with coordinate mapping $\lambda_{S_{2i}S_2}$;
  - Eq, IsEq operation, the replication dimensions γ. The operation inputs are S_1 (with coordinate mapping $\lambda_{S_1 Eq}$) and S_2 (with coordinate mapping $\lambda_{S_2 Eq}$);
- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;
  - O_{Eq}, OutputB operation, replication dimensions γ, and input Eq, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - S_1, S_2, defined in the same way as on the initial DGFR;

  - Eq_i, for $1 \leq i \leq n$, IsEq operation, the replication dimensions γ. The operation inputs are S_{1i}, with coordinate mapping $\lambda_{S_1 Eq} \circ \lambda_{S_{1i} S_1}$ and S_{2i}, with coordinate mapping $\lambda_{S_2 Eq} \circ \lambda_{S_{2i} S_2}$.
  - OK_1, IsOK operation, replication dimensions α, the input is S_1, with identity coordinate mapping;
  - OK_2, IsOK operation, replication dimensions β, the input is S_2, with identity coordinate mapping;
  - A, And operation, replication dimensions γ. The operation has $n + 2$ inputs: OK_1, OK_2, and Eq_i, for $1 \leq i \leq n$, all with identity coordinate mappings;

- Outputs
  - O_{S_1}, O_{S_2}, defined in the same way as on the original DGFR;
  - O_{Eq}, OutputB operation, replication dimensions γ, and input A, with identity coordinate mapping.

Equality of a Value and the Result of Merge operation

Let the initial DGFR be the following:

- Inputs
  - C_M, InputB operation, replication dimensions ι_{C_M};
  - S_{1i}, for $1 \leq i \leq n$, InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_2, InputS operation, replication dimensions ι_{S_2};
- Regular nodes
  - M, Merge operation, with replication dimensions α. The operation control input is C_M, with the coordinate mapping $\lambda_{C_M M}$. The operation has n data inputs, the i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i} M}$;
  - Eq, IsEq operation, the replication dimensions β. The operation inputs are M (with coordinate mapping $\lambda_{M Eq}$) and S_2 (with coordinate mapping $\lambda_{S_2 Eq}$);
- Outputs
  - O_M, OutputS operation, replication dimensions α, and input M, with identity coordinate mapping;
  - O_{Eq}, OutputB operation, replication dimensions β, and input Eq, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - M, defined in the same way as in the initial DGFR;
  - Eq_i, for $1 \leq i \leq n$, IsEq operation, and replication dimensions β. The operation inputs are S_{1i}, with coordinate mapping $\lambda_{S_{1i}M} \circ \lambda_{MEq}$, and S_2, with coordinate mapping λ_{S_2Eq};
  - OK_M, IsOK operation, replication dimensions α, the input is M, with identity coordinate mapping;
  - OK_{S_2}, IsOK operation, replication dimensions ι_{S_2}, the input is S_2, with identity coordinate mapping;
  - A, And operation, replication dimensions β. The operation has $n+2$ inputs: OK_M, with coordinate mapping λ_{MEq}, OK_{S_2}, with coordinate mapping λ_{S_2Eq}, and Eq_i, for $1 \leq i \leq n$, with identity coordinate mappings;
- Outputs
  - O_M, defined in the same way as in the initial DGFR;
  - O_{Eq}, OutputB operation, replication dimensions β, and input A, with identity coordinate mapping.

Equality of a Value and the result Tuple operation

This transformation is useful for analyzing both the IsEq and IsNeq operations. When one of the values being compared is produced by Tuple node, and the other one is not, the second comparison operand is first decomposed into the appropriate number of components (using the Proj operations), and then again assembled into the tuple.

Let the initial DGFR be the following:

- Inputs
  - C_T, InputB operation, replication dimensions ι_{C_T};
  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$, InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2j}, for $1 \leq j \leq m$ (m may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2j}}$;
- Regular nodes

  - T, Tuple operation, replication dimensions α. The operation control input is C_T, with coordinate mapping $\lambda_{C_T T}$. The operation has n data inputs; the i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}T}$;
  - S_2, any operation returning the bit string value, except for Tuple. Let the operations have m data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2}S_2}$. The operation i-th data input is S_{2i}, with coordinate mapping $\lambda_{S_{2i}S_2}$;
  - *Comp*, IsEq or IsNeq operation, the replication dimensions γ. The operation inputs are T (with coordinate mapping $\lambda_{T\,Comp}$) and S_2 (with coordinate mapping $\lambda_{S_2\,Comp}$);

- Outputs
  - O_T, OutputS operation, replication dimensions α, and input T, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;
  - O_{Comp}, OutputB operation, replication dimensions γ, and input *Comp*, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - T, S_2, defined in the same way as on the initial DGFR;
  - P_i, for $1 \leq i \leq n$, the $\mathsf{Proj}(i, n)$ operation (taking the i-th component of n-tuple), with replication dimensions β, the control input C_{S_2} (coordinate mapping $\lambda_{C_{S_2}S_2}$), and data input S_2 (identity coordinate mapping);
  - T_2, Tuple operation, replication dimensions β. The operation control input is C_{S_2} (coordinate mapping $\lambda_{C_{S_2}S_2}$). The operation has n data inputs; the i-th data input is P_i, with identity coordinate mapping;
  - *Comp*$'$, same operation as *Comp*, the replication dimensions γ. The operation inputs are T (with coordinate mapping $\lambda_{T\,Comp}$) and T_2 (with coordinate mapping $\lambda_{S_2\,Comp}$);
- Outputs

  - O_T, O_{S_2}, defined in the same way as on the original DGFR;
  - O_{Comp}, OutputB operation, replication dimensions γ, and input $Comp'$, with identity coordinate mapping.

Inequality of the Node to Itself

Let the initial DGFR be the following:

- Inputs
  - S, InputB operation; replication dimensions ι_S;
- Regular nodes
  - Neq, IsNeq operation, with replication dimensions α, and both inputs equal to S, with coordinate mapping λ_{SNeq};
- Outputs
  - O_{Neq}, OutputS operation, with replication dimensions α and input Neq, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - F, False operation, with no replication dimensions;
- Outputs
  - O_{Neq}, OutputS operation, with replication dimensions α and input F, with identity coordinate mapping.

Inequality of the Values Produced by Different Constructor-Type Operations

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;

  - S_{2j}, for $1 \leq j \leq m$ (m may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2j}}$;

- Regular nodes

  - S_1, any constructor-type operation having n data inputs and returning bit string, replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1}S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}S_1}$;
  - S_2, any constructor-type operation, which is either different from S_1, or, if S_1 is RS, the S_2 may also be RS. Let the operations have m data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2}S_2}$. The operation j-th data input is S_{2j}, with coordinate mapping $\lambda_{S_{2j}S_2}$;
  - Neq, IsNeq operation, the replication dimensions γ. The operation inputs are S_1 (with coordinate mapping λ_{S_1Neq}) and S_2 (with coordinate mapping λ_{S_2Neq});

- Outputs

  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;
  - O_{Neq}, OutputB operation, replication dimensions γ, and input Neq, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes

  - S_1, S_2, defined in the same way as on the initial DGFR;
  - OK_1, IsOK operation, replication dimensions α, the input is S_1, with identity coordinate mapping;
  - OK_2, IsOK operation, replication dimensions β, the input is S_2, with identity coordinate mapping;
  - A, And operation, replication dimensions γ. The operation has 2 inputs: OK_1 and OK_2, with coordinate mappings λ_{S_1Neq} and λ_{S_2Neq}, correspondingly;

- Outputs

  - O_{S_1}, O_{S_2}, defined in the same way as on the original DGFR;
  - O_{Neq}, OutputB operation, replication dimensions γ, and input A, with identity coordinate mapping.

Inequality of the Values Produced by Same Constructor-Type Operation

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2i}}$;
- Regular nodes
  - S_1, any constructor-type operation having n data inputs and returning bit string, replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1}S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}S_1}$;
  - S_2, same operation as for S_1. Let the operations have n data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2}S_2}$. The operation i-th data input is S_{2i}, with coordinate mapping $\lambda_{S_{2i}S_2}$;
  - Neq, IsNeq operation, the replication dimensions γ. The operation inputs are S_1 (with coordinate mapping λ_{S_1Neq}) and S_2 (with coordinate mapping λ_{S_2Neq});
- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;
  - O_{Neq}, OutputB operation, replication dimensions γ, and input Neq, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - S_1, S_2, defined in the same way as on the initial DGFR;
  - Neq_i, for $1 \leq i \leq n$, IsNeq operation, the replication dimensions γ. The operation inputs are S_{1i}, with coordinate mapping $\lambda_{S_1 Neq} \circ \lambda_{S_{1i} S_1}$ and S_{2i}, with coordinate mapping $\lambda_{S_2 Neq} \circ \lambda_{S_{2i} S_2}$.
  - OK_1, IsOK operation, replication dimensions α, the input is S_1, with identity coordinate mapping;
  - OK_2, IsOK operation, replication dimensions β, the input is S_2, with identity coordinate mapping;
  - O, Or operation, replication dimensions γ. The operation has $n+2$ inputs: OK_1, OK_2, and Neq_i, for $1 \leq i \leq n$, all with identity coordinate mappings;
- Outputs
  - O_{S_1}, O_{S_2}, defined in the same way as on the original DGFR;
  - O_{Neq}, OutputB operation, replication dimensions γ, and input O, with identity coordinate mapping.

Inequality of a Value and the Result of Merge operation

Let the initial DGFR be the following:

- Inputs
  - C_M, InputB operation, replication dimensions ι_{C_M};
  - S_{1i}, for $1 \leq i \leq n$, InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_2, InputS operation, replication dimensions ι_{S_2};
- Regular nodes
  - M, Merge operation, with replication dimensions α. The operation control input is C_M, with the coordinate mapping $\lambda_{C_M M}$. The operation has n data inputs, the i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i} M}$;
  - Neq, IsNeq operation, the replication dimensions β. The operation inputs are M (with coordinate mapping $\lambda_{M Neq}$) and S_2 (with coordinate mapping $\lambda_{S_2 Neq}$);
- Outputs

  - O_M, OutputS operation, replication dimensions α, and input M, with identity coordinate mapping;
  - O_{Neq}, OutputB operation, replication dimensions β, and input Neq, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - M, defined in the same way as in the initial DGFR;
  - Neq_i, for $1 \leq i \leq n$, IsNeq operation, and replication dimensions β. The operation inputs are S_{1i}, with coordinate mapping $\lambda_{S_{1i}M} \circ \lambda_{MNeq}$, and S_2, with coordinate mapping λ_{S_2Neq};
  - OK_M, IsOK operation, replication dimensions α, the input is M, with identity coordinate mapping;
  - OK_{S_2}, IsOK operation, replication dimensions ι_{S_2}, the input is S_2, with identity coordinate mapping;
  - O, Or operation, replication dimensions β. The operation has $n+2$ inputs: OK_M, with coordinate mapping λ_{MNeq}, OK_{S_2}, with coordinate mapping λ_{S_2Neq}, and Neq_i, for $1 \leq i \leq n$, with identity coordinate mappings;
- Outputs
  - O_M, defined in the same way as in the initial DGFR;
  - O_{Neq}, OutputB operation, replication dimensions β, and input O, with identity coordinate mapping.

Moving the IsOK over the Constructor-type Node

Let the initial DGFR be the following:

- Inputs
  - C_S, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - S_i, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions ι_{S_i};
- Regular nodes

  - S, any constructor-type operation having n data inputs and returning bit string, replication dimensions α. The operation control input is C_S, with coordinate mapping $\lambda_{C_S S}$. The operation i-th data input is S_i, with coordinate mapping $\lambda_{S_i S}$;
  - OK, IsOK operation, the replication dimensions γ. The operation input is S (with coordinate mapping $\lambda_{S OK}$);

- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping;
  - O_{OK}, OutputB operation, replication dimensions γ, and input OK, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - S, defined in the same way as on the initial DGFR;
  - OK_i, for $1 \leq i \leq n$, IsOK operation, the replication dimensions ι_{S_i}. The operation inputs are S_i, with identity coordinate mapping;
  - OK', And operation, replication dimensions γ. The operation has $n+1$ inputs: C_S (coordinate mapping $\lambda_{C_S S}$) and OK_i, for $1 \leq i \leq n$, all with coordinate mapping $\lambda_{S_i S}$;
- Outputs
  - O_S, defined in the same way as on the original DGFR;
  - O_{OK}, OutputB operation, replication dimensions γ, and input OK', with identity coordinate mapping.

Simplifying Computations Resulting in Error

This group of transformations relies on the fact that the constructor-type operations (the nodes which compose the value out of components, like Tuple, in contrast to the operations extracting the value — e.g. Proj) tag the result with the type identifier, and certain operations expect the value received on inputs ports, to be of the certain type.

The constructor type operations, referred below, are: Error, RS, Nonce, Const, Keypair, PubKey, SigVer, VerKey, SymKey, PubEnc, SymEnc, PubEncZ, SymEncZ, Signature, Tuple, and Secret.

Argument of Incompatible Type for the Bit String Node

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - C_{S_2}, InputB operation, replication dimensions $\iota_{C_{S_2}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2j}, for $1 \leq j \leq m$) (m may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2j}}$;
- Regular nodes
  - S_1, a constructor-type operation, returning the value of a type different from what is expected by the port p of S_2. The compatibility table is supplied below. Let the node have n data inputs and replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1}S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}S_1}$;
  - S_2, the operation checking the type of the input port p. Let the operation have $m+1$ data inputs and replication dimensions β. The operation control input is C_{S_2}, with coordinate mapping $\lambda_{C_{S_2}S_2}$. Let the k be the input, corresponding to input port p. The operation j-th data input is either S_{2j} (if $j < k$), with coordinate mapping $\lambda_{S_{2j}S_2}$, or S_{2j}, with coordinate mapping $\lambda_{S_{2j+1}S_2}$(if $j < k$), or S_1, with coordinate mapping $\lambda_{S_1S_2}$ (if $j = k$);
- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions β, and input S_2, with identity coordinate mapping;

The operations checking the types of the arguments, and the corresponding compatible types are listed in the table 2.

If the S_1 has the type incompatible with what the port p of the S_2 expects, the initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes

Table 2: Acceptable constructor types for operations arguments

Operation	Port	Acceptable Constructor
Nonce, Keypair, SigVer, SymKey, PubEnc, SymEnc, Signature, PubEncZ, SymEncZ	Random coins	RS
PubEnc, PubEncZ	Key	PubKey
SymEnc, SymEncZ, SymDec	Key	SymKey
Signature	Key	SigVer
PubDec	Key	PubKey
Proj(i, n)	Data	Tuple(n)
Any bit string operation	Any port	Different from Error
Any bit string operation	Control	Different from False

  - S_1, defined in the same way as on the initial DGFR;
  - E, Error operation, no replication dimensions;

- Outputs
  - O_{S_1}, defined in the same way as on the original DGFR;
  - O_{S_2}, OutputS operation, replication dimensions β, input E, with identity coordinate mapping.

Argument of Incompatible Type for the Bit String-to-Boolean Node

Let the initial DGFR be the following:

- Inputs
  - C_{S_1}, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
  - S_{1i}, for $1 \leq i \leq n$ (n may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{1i}}$;
  - S_{2j}, for $1 \leq j \leq m$) (m may be equal to zero!), InputS operation, replication dimensions $\iota_{S_{2j}}$;

- Regular nodes
  - S_1, a constructor-type operation, returning the value of a type different from what is expected by the port p of S_2. The compatibility table is supplied below. Let the node have n data inputs and replication dimensions α. The operation control input is C_{S_1}, with coordinate mapping $\lambda_{C_{S_1}S_1}$. The operation i-th data input is S_{1i}, with coordinate mapping $\lambda_{S_{1i}S_1}$;

  - S_2, the bit string-to-boolean operation checking the type of the input port p. Let the operation have $m+1$ data inputs and replication dimensions β. Let the k be the input, corresponding to input port p. The operation j-th data input is either S_{2j} (if $j < k$), with coordinate mapping $\lambda_{S_{2j}S_2}$, or S_{2j}, with coordinate mapping $\lambda_{S_{2j+1}S_2}$(if $j < k$), or S_1, with coordinate mapping $\lambda_{S_1S_2}$ (if $j = k$);

- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputB operation, replication dimensions β, and input S_2, with identity coordinate mapping;

The operations checking the types of the arguments, and the corresponding compatible types are listed in the table 3.

Table 3: Acceptable constructor types for operations arguments

Operation	Port	Acceptable Constructor
TestSig, TestSigP	Key	VerKey
Any operation	Any port	Different from Error

If the S_1 has the type incompatible with what the port p of the S_2 expects, the initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - S_1, defined in the same way as on the initial DGFR;
  - F, False operation, no replication dimensions;
- Outputs
  - O_{S_1}, defined in the same way as on the original DGFR;
  - O_{S_2}, OutputB operation, replication dimensions β, input F, with identity coordinate mapping.

IfDef **Node**

There are four possible cases when the IfDef (same applies to CfDef) node can be transformed:

- The check-data input pair with the check input equal to false can be removed;

- The check-data input pair with the data input equal to Error (or False in case of CfDef) can be removed;
- If the IfDef has only a single pair of check-data inputs, it can be replaced with Id;
- Finally, if the node has no check-data input pairs, its result is Error (False in case of CfDef).

The formal definition of these transformations is similar to the transformations described above, so we omit it here.

Transformations Based on Boolean Arithmetic

Simplifying Boolean Operations Having a Single Input

An And or Or node, having the single input, can be replaced with its input. A Longor operation, with no replication dimensions contacted can also be replaced with its input.

Formally, let the initial DGFR be the following:

- Inputs
  - B, InputB operation, replication dimensions ι_B;
- Regular nodes
  - A, And or Or or Longor (with no contacted dimensions) operation; replication dimensions α; the input is B, with coordinate mapping λ_{BA};
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;
- No regular nodes
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input B, with coordinate mapping λ_{BA};

Simplifying Boolean Operations Always Evaluating to a Constant

In certain cases the boolean operation always evaluates to constant, irrespective of its inputs. In these cases the operation can be replaced with the constant it evaluates to. These cases are:

- And operation evaluates to false if one of its inputs is False;
- Or operation evaluates to true if one of its inputs is True;
- Longor operation evaluates to true if its input is True;
- DimEq(i, i) and DimNeq(i, i) operations (comparing the coordinate with itself) evaluate to true and false, respectively;
- IsOK operation evaluate to false, if its input is Error.

Let us illustrate the transformation on the example of And. Formally, let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - B, False operation, replication dimensions β;
  - A, And operation with replication dimensions α; The operation has $n + 1$ inputs: B (coordinate mapping λ_{BA}) and, for $1 \leq i \leq n$, B_i (coordinate mapping λ_{B_iA});
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - B, False operation, no replication dimensions;
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input B, with coordinate mapping λ_{BA};

Simplifying a Sequence of And or Or Operations

Let the initial DGFR be the following:

- Inputs
  - B_{1i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{B_{1i}}$;
  - B_{2j}, for $1 \leq i \leq m$, InputB operation, replication dimensions $\iota_{B_{2j}}$;
- Regular nodes
  - A_1, And or Or operation with replication dimensions α_1; The operation has n inputs: for $1 \leq i \leq n$, B_{1i} (coordinate mapping $\lambda_{B_{1i}A_1}$);
  - A_2, same operation as A_1, replication dimensions β; The operation has $m+1$ inputs: A_1 (coordinate mapping $\lambda_{A_1A_2}$) and, for $1 \leq j \leq m$, B_{2j} (coordinate mapping $\lambda_{B_{2j}A_2}$);
- Outputs
  - O_{A_1}, OutputB operation, replication dimensions α, and input A_1, with identity coordinate mapping;
  - O_{A_2}, OutputB operation, replication dimensions β, and input A_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - A_1, defined in the same way as on the initial DGFR;
  - A'_2, same operation as A_1, replication dimensions β; The operation has $m+n$ inputs: for $1 \leq i \leq n$, B_{2i} (coordinate mapping $\lambda_{B_{1i}A_1}) \circ \lambda_{A_1A_2}$) and, for $1 \leq j \leq m$, B_{2j} (coordinate mapping $\lambda_{B_{2j}A_2}$);
- Outputs
  - O_{A_1}, defined in the same way as on the initial DGFR;
  - O_{A_2}, OutputB operation, replication dimensions β, and input A'_2, with identity coordinate mapping;

Simplifying a Sequence of Longor Operations

Let the initial DGFR be the following:

- Inputs
  - B_1, InputB operation, replication dimensions ι_{B_1};
- Regular nodes
  - LO_1, Longor operation with replication dimensions α. The operation input is B_1, with coordinate mapping $\lambda_{B_1 LO_1}$ and contracted dimensions α';
  - LO_2, Longor operation with replication dimensions β. The operation input is LO_1, with coordinate mapping $\lambda_{LO_1 LO_2}$ and contracted dimensions β';
- Outputs
  - O_{LO_1}, OutputB operation, replication dimensions α, and input LO_1, with identity coordinate mapping;
  - O_{LO_2}, OutputB operation, replication dimensions β, and input LO_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - LO_1, defined in the same way as on the initial DGFR;
  - LO'_2, Longor operation with replication dimensions β. The operation input is B_1, with coordinate mapping $\lambda_{B_1 LO_1} \circ \lambda_{LO_1 LO_2}$ and contracted dimensions $\alpha' + \beta'$ (pointwise addition);
- Outputs
  - O_{LO_1}, defined in the same way as on the initial DGFR;
  - O_{LO_2}, OutputB operation, replication dimensions β, and input LO'_2, with identity coordinate mapping;

Simplifying a Sequence of Or and Longor Operations

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - O, Or operation with replication dimensions α; The operation has n inputs: for $1 \leq i \leq n$, B_i (coordinate mapping λ_{B_iO});
  - LO, Longor operation with replication dimensions β. The operation input is O, with coordinate mapping λ_{OLO} and contracted dimensions β';
- Outputs
  - O_{LO}, OutputB operation, replication dimensions β, and input LO, with identity coordinate mapping;

Let the initial DGFR be the following:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - LO'_i, for $1 \leq i \leq n$, Longor operation with replication dimensions β. The operation input is B_i (coordinate mapping λ_{B_iO}), contracted dimensions β';
  - O', Or operation with replication dimensions β; The operation has n inputs: for $1 \leq i \leq n$, LO'_i, all with identity coordinate mapping;
- Outputs
  - O_{LO}, OutputB operation, replication dimensions β, and input O', with identity coordinate mapping;

Distributing the Inputs of the Or Node

Let the initial DGFR be the following:

- Inputs
  - A_i, for $1 \leq i \leq a$, InputB operation, replication dimensions α_i;
  - B_{ij}, for $1 \leq i \leq c$, $1 \leq j \leq b_i$ (b_i has a separate value for each $1 \leq i \leq c$), InputB operation, replication dimensions β_{ij};

- Regular nodes
  - C_i, for $1 \leq i \leq c$, And operation, replication dimensions γ_i. The node has $a+b_i$ inputs: A_k (for $1 \leq k \leq a$), with coordinate mapping $\lambda_{A_k C_i}$, and B_{ij} (for $1 \leq j \leq b_i$), with coordinate mapping $\lambda_{B_{ij} C_i}$;
  - D, Or operation, replication dimensions δ. The node has c inputs - the i-th input (for $1 \leq i \leq c$) is C_i, with coordinate mapping $\lambda_{C_i D}$;
- Outputs
  - O_{C_i}, for $1 \leq i \leq c$, OutputB operation, replication dimensions γ_i. The input is C_i, with identity coordinate mapping;
  - O_D, OutputB operation, replication dimensions δ. The input is D, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as on the initial DGFR;
- Regular nodes
  - C_i, defined in the same way as on the original DGFR;
  - $NewAnd_i$, $1 \leq i \leq c$, the operation is *And* with b_i inputs, the number of dimensions is γ_i. The inputs are B_{ij}, for $1 \leq j \leq b_i$, with dimension mapping $\lambda_{B_{ij} C_i}$;
  - *NewOr*, the operation is *Or* with c inputs, the number of dimensions is δ, the inputs are $NewAnd_i$, $1 \leq i \leq c$, with dimension mapping $\lambda_{C_i D}$;
  - $\dot{D}$, the operation is *And* with $a+1$ inputs, replication dimensions is δ, the inputs are
    * A_i, $1 \leq i \leq a$, with coordinate mapping $\lambda_{A_i C_1} \circ \lambda_{C_1 D}$;
    * *NewOr*, with identity coordinate mapping.
- Outputs
  - O_{C_i}, defined in the same way as on the initial DGFR;
  - O_D, OutputB operation, replication dimensions δ. The input is $\dot{D}$, with identity coordinate mapping;

The transformation of the graph fragment with $a = 2$, $c = 2$, $b_1 = 1$, $b_2 = 2$, and all dimensionsion mappings empty (mapping every $D \in Dim$ to 0) is illustrated in Figure 12.

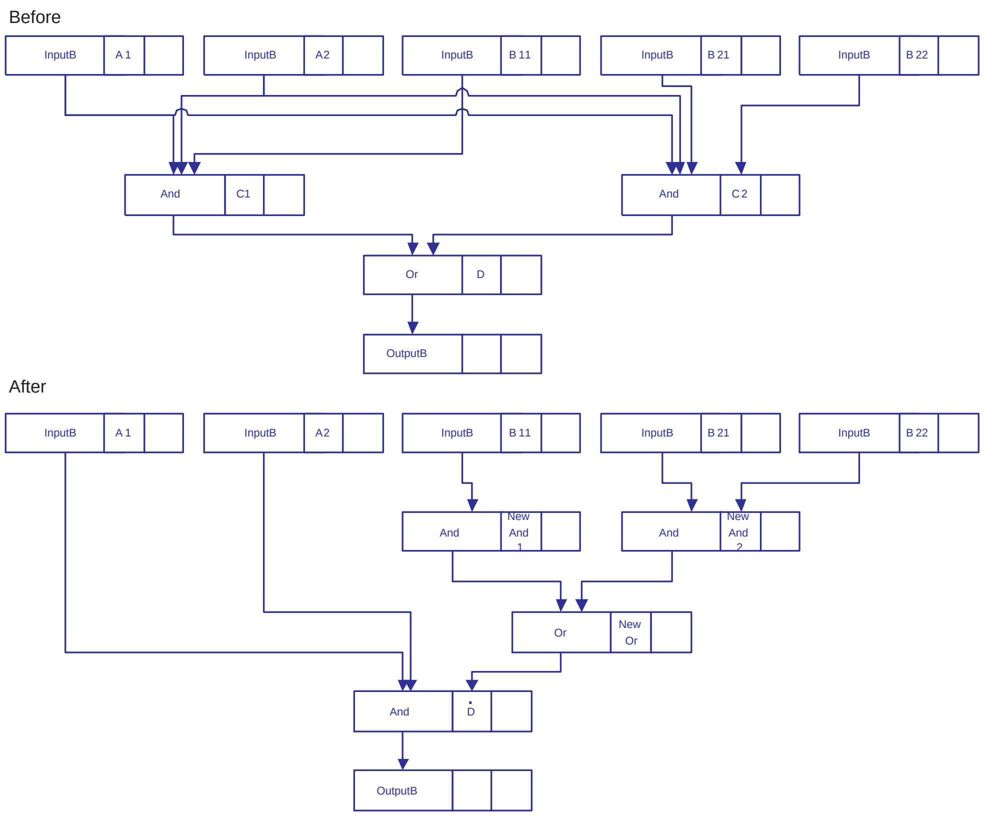

Figure 12: Example of Distribute Or transformation

Distributing the Inputs of the Longor Node

If the input to the Longor-node is an And-node, and some if its inputs has the same or less number of coordinates than Longor-node, these inputs could be processed in the new And node, after the Longor node computation is performed.

Formally, it means that for all $1 \leq i \leq a$ the α_i does not contain coordinates which are not present in δ.

Let the initial DGFR be the following:

- Inputs
  - A_i, for $1 \leq i \leq a$, InputB operation, replication dimensions α_i;
  - B_j, for $1 \leq j \leq b$, InputB operation, replication dimensions β_j;
- Regular nodes
  - C, And operation, replication dimensions γ. The node has $a + b$ inputs: A_i (for $1 \leq i \leq a$), with coordinate mapping λ_{A_iC}, and B_j (for $1 \leq j \leq b$), with coordinate mapping λ_{B_jC};
  - D, Longor operation, replication dimensions δ. The node input is C, with coordinate mapping λ_{CD} and contracted dimensions δ';
- Outputs
  - O_C, OutputB operation, replication dimensions γ. The input is C, with identity coordinate mapping;
  - O_D, OutputB operation, replication dimensions δ. The input is D, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- Inputs - same as on the initial DGFR;
- Regular nodes
  - C, defined in the same way as on the original DGFR;
  - $NewAnd$, the operation is And with b inputs, the number of dimensions is γ. The inputs are B_j, for $1 \leq j \leq b$, with dimension mapping λ_{B_jC};
  - $NewLO$, Longor operation, replication dimensions δ. The node input is $NewAnd$, with coordinate mapping λ_{CD} and contracted dimensions δ';
  - $\dot{D}$, the operation is And with $a + 1$ inputs, replication dimensions is δ, the inputs are

        * A_i, $1 \le i \le a$, with dimension mapping $\lambda_{A_i C_1} \circ \lambda_{C_1 D}$;
        * $NewLO$, with identity coordinate mapping.

- Outputs
    - O_{C_i}, defined in the same way as on the initial DGFR;
    - O_D, OutputB operation, replication dimensions δ. The input is $\dot{D}$, with identity coordinate mapping;

Removing Duplicate Copies of Operations

The transformations in this section are based on the fact that certain nodes will always return the same value, so it is possible to group them. For instance, two nodes (except for RS and Receive), having the same inputs, will return the same value.

Superfluous Dimension of an Node with Control Input

Let the initial DGFR be the following:

- Inputs
    - C_S, InputB operation, replication dimensions $\iota_{C_{S_1}}$;
    - S_i, for $1 \le i \le n$ (n may be equal to zero!), InputS operation, replication dimensions ι_{S_i};
- Regular nodes
    - S, any operation, except for RS and Receive, having n data inputs and returning bit string, replication dimensions α. The operation control input is C_S, with coordinate mapping $\lambda_{C_S S}$. The operation i-th data input is S_i, with coordinate mapping $\lambda_{S_i S}$;
- Outputs
    - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping;

If the replication dimensions α contains a coordinate, which has no corresponding coordinate (through the $\iota_{S_i}^{-1}$) in any of the operation data inputs S_i, then the initial DGFR can be replaced with the following one:

- Inputs — the same as in the initial DGFR;
- Regular nodes

  - LO, Longor operation, replication dimensions α', input C_S, with coordinate mapping $\lambda_{C_S S}$, and contracted dimensions β;
  - S', same operation as for S, replication dimensions α'. The operation control input is LO, with identity coordinate mapping. The operation i-th data input is S_i, with coordinate mapping $\lambda_{S_i S}$;
  - ID, Id operation, replication dimensions α, control input input C_S, with coordinate mapping $\lambda_{C_S S}$, and data input S', with identity coordinate mapping;

- Outputs
  - O_S, OutputS operation, replication dimensions α, and input ID, with identity coordinate mapping;

The replication dimensions α' includes only those replication dimensions given by α, which have the corresponding coordinate (through the $\iota_{S_i}^{-1}$) in at least on of the S_i (for $1 \leq i \leq n$). The rest of coordinates given by α are contracted in β.

Superfluous Dimension of an Node with no Control Input

Let the initial DGFR be the following:

- Inputs
  - X_i, for $1 \leq i \leq n$, InputS or InputB operation, replication dimensions ι_{S_i};

- Regular nodes
  - X, any operation, except for RS and Receive, having n inputs and replication dimensions α. The operation i-th input is X_i, with coordinate mapping $\lambda_{X_i X}$;

- Outputs
  - O_X, OutputS or OutputB operation, replication dimensions α, and input X, with identity coordinate mapping;

If the replication dimensions α contains a coordinate, which has no corresponding coordinate (through the $\iota_{X_i}^{-1}$) in any of the operation inputs X_i, then the initial DGFR can be replaced with the following one:

- Inputs — the same as in the initial DGFR;

- Regular nodes

  - X', same operation as X, replication dimensions α'. The operation i-th input is X_i, with coordinate mapping $\lambda_{X_i X}$;

- Outputs
  - O_X, same operation as the corresponding node on the original DGFR, replication dimensions α, and input X, with identity coordinate mapping;

The replication dimensions α' includes only those replication dimensions given by α, which have the corresponding coordinate (through the $\iota_{S_i}^{-1}$) in at least on of the X_i (for $1 \leq i \leq n$).

Superfluous Dimension of an Node with no Inputs

Let the initial DGFR be the following:

- No inputs
- Regular nodes
  - X, an operation with no inputs and replication dimensions α;
- Outputs
  - O_X, OutputS or OutputB operation, replication dimensions α, and input X, with identity coordinate mapping;

If the replication dimensions α contains a coordinate not influencing the operation result, this coordinate can be removed. The coordinates α' influencing the results of the operations with no inputs are given in the table 4.

Table 4: Replication dimensions influencing the result of the operation with no inputs

Operation	Replication Dimensions α'
True, False, Const(i), Error	$\alpha(d) = 0$
DimEq(d_1, i, j),DimNeq(d_1, i, j)	$\alpha(d_1) = 2$ $\alpha(d \neq d_1) = 0$

If α is different from α', then the initial DGFR can be replaced with the following one:

- No inputs
- Regular nodes

  - X, an operation with no inputs and replication dimensions α';

- Outputs
  - O_X, same operation as the corresponding node on the original DGFR, replication dimensions α, and input X, with identity coordinate mapping;

Two nodes with Identical Inputs

Let the initial DGFR be the following:

- Inputs
  - X_i, for $1 \leq i \leq n$, InputS or InputB operation, replication dimensions ι_{S_i};
- Regular nodes
  - X, any operation, except for RS and Receive, having n inputs and replication dimensions α. The operation i-th input is X_i, with coordinate mapping $\lambda_{X_i X}$;
  - X', same operation as X, replication dimensions α. The operation i-th input is X_i, with coordinate mapping $\lambda_{X_i X}$;
- Outputs
  - O_X, OutputS or OutputB operation, replication dimensions α, and input X, with identity coordinate mapping;
  - $O_{X'}$, OutputS or OutputB operation, replication dimensions α, and input X', with identity coordinate mapping;

The initial DGFR can be replaced with the following one:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - X, defined as in the original DGFR;
- Outputs
  - O_X, defined as in the original DGFR;
  - $O_{X'}$, same operation as the corresponding node on the original DGFR, replication dimensions α, and input X, with identity coordinate mapping;

Two nodes with Identical Data Inputs, but Different Control Input

Let the initial DGFR be the following:

- Inputs
  - C_1, InputB operation, replication dimensions ι_{C_1};
  - C_2, InputB operation, replication dimensions ι_{C_2};
  - S_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{S_i};
- Regular nodes
  - S_1, any operation, except for RS and Receive, having the control and n data inputs and replication dimensions α. The operation control input is C_1, with coordinate mapping $\lambda_{C_1S_1}$. The operation i-th data input is S_i, with coordinate mapping λ_{S_iS};
  - S_2, same operation as S_1, replication dimensions α. The operation control input is C_2, with coordinate mapping $\lambda_{C_2S_2}$. The operation i-th data input is S_i, with coordinate mapping λ_{S_iS};
- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input S_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions α, and input S_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following one:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - O, Or operation, replication dimensions α. The node has two inputs: C_1 with coordinate mapping $\lambda_{C_1S_1}$ and C_2 with coordinate mapping $\lambda_{C_2S_2}$;
  - S, same operation as S_1, replication dimensions α. The operation control input is O, with identity coordinate mapping. The operation i-th data input is S_i, with coordinate mapping λ_{S_iS};
  - ID_1, Id operation, replication dimensions α, control input C_1 with coordinate mapping $\lambda_{C_1S_1}$ and data input S, with identity coordinate mapping;
  - ID_2, Id operation, replication dimensions α, control input C_2 with coordinate mapping $\lambda_{C_2S_2}$ and data input S, with identity coordinate mapping;

- Outputs
  - O_{S_1}, OutputS operation, replication dimensions α, and input ID_1, with identity coordinate mapping;
  - O_{S_2}, OutputS operation, replication dimensions α, and input ID_2, with identity coordinate mapping;

Two Nodes With no Inputs

Let the initial DGFR be the following:

- No Inputs
- Regular nodes
  - E_1, Error operation, replication dimensions α;
  - E_2, Error operation, replication dimensions β;
- Outputs
  - O_{E_1}, OutputS operation, replication dimensions α, and input E_1, with identity coordinate mapping;
  - O_{E_2}, OutputS operation, replication dimensions β, and input E_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following:

- No Inputs
- Regular nodes
  - E, Error operation, no replication dimensions;
- Outputs
  - O_{E_1}, OutputS operation, replication dimensions α, and input E, with identity coordinate mapping;
  - O_{E_2}, OutputS operation, replication dimensions β, and input E, with identity coordinate mapping;

The same transformation can be performed to the nodes with the False and True operations.

Propagating Control Dependencies

Propagating Control Dependency to an IfDef-Node

Let the initial DGFR be the following:

- Inputs
  - C_I, InputB operation and replication dimensions ι_{C_I};
  - $Check_i$, for $1 \leq i \leq n+1$, InputB operation, the replication dimensions are given by ι_{CD_i};
  - $Data_i$, for $1 \leq i \leq n$, InputS operation, the replication dimensions are given by ι_{CD_i};
  - C_{OP}, InputB operation and replication dimensions $\iota_{C_{OP}}$;
  - Arg_j, for $1 \leq j \leq m$, *opinputs* operation, and replication dimensions ι_{Arg_j};
- Regular nodes
  - OP, any operation returning bit string value and having one control and m data inputs; the replication dimensions are given by $\iota_{CD_{n+1}}$; the control input is C_{OP}, with coordinate mapping $\lambda_{C_{OP}OP}$; the j-th input $(1 \leq j \leq m)$ is Arg_j with coordinate mapping λ_{Arg_jOP};
  - I, IfDef operation; the replication dimensions are given by α; the control input is C_I, with coordinate mapping λ_{C_II}; the node has $n+1$ pairs of check-data inputs - the for $1 \leq i \leq n$ i-th pair is $Check_i$ and $Data_i$, with the coordinate mapping λ_{CD_iI} and contracted dimensions γ_i; The $n+1$-th pair comprises the check input $Check_{n+1}$ and data input OP, with the coordinate mapping $\lambda_{CD_{n+1}I}$ and contracted dimensions γ_{n+1};
- Outputs
  - O_{OP}, OutputS operation, replication dimension $\iota_{CD_{n+1}}$, the node input is OP, with identity coordinate mapping.
  - O_I, OutputS operation, replication dimensions α,the node input is I, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs - same as on the initial DGFR;
- Regular nodes
  - OP, same as on the initial DGFR;

  - $NewCheck_{n+1}$, And operation, replication dimensions $\iota_{CD_{n+1}}$. The operation inputs are $Check_{n+1}$ (identity coordinate mapping) and C_{OP} ($\lambda_{C_{OP}OP}$ coordinate mapping);
  - $NewI$, IfDef operation; the replication dimensions are given by α; the control input is C_I, with coordinate mapping $\lambda_{C_I I}$; the node has $n+1$ pairs of check-data inputs - the for $1 \leq i \leq n$ i-th pair is $Check_i$ and $Data_i$, with the coordinate mapping $\lambda_{CD_i I}$ and contracted dimensions γ_i; The $n+1$-th pair comprises the check input $NewCheck_{n+1}$ and data input OP, with the coordinate mapping $\lambda_{CD_{n+1}I}$ and contracted dimensions γ_{n+1};

- Outputs
  - O_{OP}, OutputS operation, replication dimension β, the node input is OP, with identity coordinate mapping.
  - O_I, OutputS operation, replication dimensions α,the node input is $NewI$, with identity coordinate mapping;

Propagating Control Dependency to a Node with Control Dependency

Let the initial DGFR be the following:

- Inputs
  - C_{OP_1}, InputB operation and replication dimensions $\iota_{C_{OP_1}}$;
  - C_{OP_2}, InputB operation and replication dimensions $\iota_{C_{OP_2}}$;
  - Arg_{1i}, for $1 \leq i \leq n$, *opinputs* operation, and replication dimensions $\iota_{Arg_{1i}}$;
  - Arg_{2j}, for $1 \leq j \leq m$, *opinputs* operation, and replication dimensions $\iota_{Arg_{2j}}$;

- Regular nodes
  - *OP₁*, any operation returning bit string value and having one control and n data inputs; the replication dimensions α; the control input is C_{OP_1}, with coordinate mapping $\lambda_{C_{OP_1}OP_1}$; the i-th input $(1 \leq i \leq m)$ is Arg_{1i} with coordinate mapping $\lambda_{Arg_{1i}OP_1}$;
  - *OP₂*, any operation returning bit string value and having one control and $m+1$ data inputs; the replication dimensions *beta*; the control input is C_{OP_2}, with coordinate mapping $\lambda_{C_{OP_2}OP_2}$. The j-th data input is: for a single $j = k$, *OP₂* (coordinate mapping $\lambda OP_1 OP_2$); for all $j < k$, Arg_{2j} with coordinate mapping $\lambda_{Arg_{2j}OP_1}$; for all $j > k$, Arg_{2j+1} with coordinate mapping $\lambda_{Arg_{2j+1}OP_1}$;

- Outputs
  - O_{OP_1}, OutputS operation, replication dimension α, the node input is OP_1, with identity coordinate mapping;
  - O_{OP_2}, OutputS operation, replication dimension β, the node input is OP_2, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs - same as on the initial DGFR;
- Regular nodes
  - *OP₁*, same as on the initial DGFR;
  - $NewC_{S_2}$, And operation, replication dimensions β. The operation inputs are C_{OP_1} ($\lambda_{C_{OP_1}OP_1} \circ \lambda OP_1 OP_2$ coordinate mapping) and C_{OP_2} ($\lambda_{C_{OP_2}OP_2}$ coordinate mapping);
  - $NewOP_2$, same operation as OP_2; the replication dimensions *beta*; the control input is $NewC_{S_2}$, with identity coordinate mapping. The data inputs are the same as for the OP_2 node.
- Outputs
  - O_{OP_1}, same as on the initial DGFR;
  - O_{OP_2}, OutputS operation, replication dimension β, the node input is $NewOP_2$, with identity coordinate mapping.

Propagating Control Dependency to a Node without Control Dependency

Let the initial DGFR be the following:

- Inputs
  - C_{OP_1}, InputB operation and replication dimensions $\iota_{C_{OP_1}}$;
  - Arg_{1i}, for $1 \leq i \leq n$, *opinputs* operation, and replication dimensions $\iota_{Arg_{1i}}$;
  - Arg_{2j}, for $1 \leq j \leq m$, *opinputs* operation, and replication dimensions $\iota_{Arg_{2j}}$;
- Regular nodes
  - OP_1, any operation returning bit string value and having one control and n data inputs; the replication dimensions α; the control input is C_{OP_1}, with coordinate mapping $\lambda_{C_{OP_1}OP_1}$; the i-th input $(1 \leq i \leq m)$ is Arg_{1i} with coordinate mapping $\lambda_{Arg_{1i}OP_1}$;

  - OP_2, any operation returning boolean value and having $m+1$ data inputs; the replication dimensions *beta*; The j-th data input is: for a single $j = k$, OP_2 (coordinate mapping $\lambda OP_1 OP_2$); for all $j < k$, Arg_{2j} with coordinate mapping $\lambda_{Arg_{2j}OP_1}$; for all $j > k$, Arg_{2j+1} with coordinate mapping $\lambda_{Arg_{2j+1}OP_1}$;

- Outputs
  - O_{OP_1}, OutputS operation, replication dimension α, the node input is OP_1, with identity coordinate mapping;
  - O_{OP_2}, OutputB operation, replication dimension β, the node input is OP_2, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs - same as on the initial DGFR;
- Regular nodes
  - OP_1, OP_2, same as on the initial DGFR;
  - $NewOP2_{S_2}$, And operation, replication dimensions β. The operation inputs are C_{OP_1} ($\lambda_{C_{OP_1}OP_1} \circ \lambda OP_1 OP_2$ coordinate mapping) and OP_2 (identity coordinate mapping);
- Outputs
  - O_{OP_1}, same as on the initial DGFR;
  - O_{OP_2}, OutputS operation, replication dimension β, the node input is $NewOP_2$, with identity coordinate mapping.

Transformations Specific to IfDef Operations

Combining two IfDef Operations

Let the initial DGFR be the following:

- Inputs
  - C_{I_1}, InputB operation and replication dimensions $\iota_{C_{I_1}}$;
  - $Check_{1i}$, for $1 \leq i \leq n$, InputB operation, the replication dimensions are given by $\iota_{CD_{1i}}$;
  - $Data_{1i}$, for $1 \leq i \leq n$, InputS operation, the replication dimensions are given by $\iota_{CD_{1i}}$;
  - $Check_{2j}$, for $1 \leq j \leq m+1$, InputB operation, the replication dimensions are given by $\iota_{CD_{2j}}$;

  - $Data_{2j}$, for $1 \leq j \leq m$, InputS operation, the replication dimensions are given by $\iota_{CD_{2j}}$;

- Regular nodes
  - I_1, IfDef operation; the replication dimensions are given by $\iota_{CD_{2m+1}}$; the control input is C_{I_1}, with coordinate mapping $\lambda_{C_{I_1}I_1}$; the node has n pairs of check-data inputs - for $1 \leq i \leq n$ i-th pair is $Check_{1i}$ and $Data_{1i}$, with the coordinate mapping $\lambda_{CD_{1i}I_1}$ and contracted dimensions γ_{1i};
  - I_2, IfDef operation; the replication dimensions β; the control input is C_{I_2}, with coordinate mapping $\lambda_{C_{I_2}I_2}$; the node has $m+1$ pairs of check-data inputs - for $1 \leq j \leq m$ j-th pair is $Check_{2j}$ and $Data_{2j}$, with the coordinate mapping $\lambda_{CD_{2j}I_2}$ and contracted dimensions γ_{2j}; The m+1-th pair comprises the check input $Check_{m+1}$ and data input I_1, with the coordinate mapping $\lambda_{CD_{m+1}I}$ and contracted dimensions γ_{2m+1};

- Outputs
  - O_{I_1}, OutputS operation, replication dimensions $\iota_{CD_{2m+1}}$, the node input is I_1, with identity coordinate mapping;
  - O_{I_2}, OutputS operation, replication dimensions β, the node input is I_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs - same as on the original DGFR;

- Regular nodes
  - I_1, same as on the original DGFR;
  - $NewI_2$, IfDef operation; the replication dimensions β; the control input is C_{I_2}, with coordinate mapping $\lambda_{C_{I_2}I_2}$; the node has $m+n$ pairs of check-data inputs - for $1 \leq j \leq m$ the j-th pair is $Check_{2j}$ and $Data_{2j}$, with the coordinate mapping $\lambda_{CD_{2j}I_2}$ and contracted dimensions γ_{2j}; for $m+1 \leq j \leq m+n$ the j-th pair is $Check_{1j-m}$ and $Data_{1j-m}$, with the coordinate mapping $\lambda_{CD_{1j-m}I_1} \circ \lambda_{CD_{m+1}I}$ and contracted dimensions $\gamma_{1j-m} + \gamma_{2m+1}$ (pointwise addition);

- Outputs
  - O_{I_1}, same as on the original DGFR;
  - O_{I_2}, OutputS operation, replication dimensions β, the node input is $NewI_2$, with identity coordinate mapping;

CfDef **Decomposition**

Let the initial DGFR be the following:

- Inputs
  - C_I, InputB operation and replication dimensions ι_{C_I};
  - *Check*$_i$, for $1 \leq i \leq n$, InputB operation, the replication dimensions are given by ι_{CD_i};
  - *Data*$_i$, for $1 \leq i \leq n$, InputB operation, the replication dimensions are given by ι_{CD_i};
- Regular nodes
  - I, CfDef operation; the replication dimensions are given by α; the control input is C_I, with coordinate mapping $\lambda_{C_I I}$; the node has n pairs of check-data inputs - the i-th pair is *Check*$_i$ and *Data*$_i$, with the coordinate mapping $\lambda_{CD_i I}$ and contracted dimensions γ_i;
- Outputs
  - O_I, OutputS operation, replication dimensions α,the node input is I, with identity coordinate mapping.

The initial DGFR can be replaced with the following:

- Inputs — the same as in the initial DGFR;
- Regular nodes
  - A_i, for $1 \leq i \leq n$, And operation, replication dimensions ι_{CD_i}. The operation inputs are *Check*$_i$ and *Data*$_i$, with identity coordinate mappings;
  - LO_i, for $1 \leq i \leq n$, Longor operation, replication dimensions α. The operation input is A_i, with the coordinate mapping $\lambda_{CD_i I}$, and contracted dimensions γ_i;
  - O, Or operation, replication dimensions α. The operation has n inputs; the i-th input is LO_i, with identity coordinate mapping;
  - I', And operation, replication dimensions α. The operation has two inputs: O, with identity coordinate mapping, and C_I, with coordinate mapping $\lambda_{C_I I}$;
- Outputs
  - O_I, OutputS operation, replication dimensions α,the node input is I', with identity coordinate mapping.

Moving Operations over IfDef

Let the initial DGFR be the following:

- Inputs
  - C_I, InputB operation and replication dimensions ι_{C_I};
  - $Check_i$, for $1 \leq i \leq n$, InputB operation, the replication dimensions are given by ι_{CD_i};
  - $Data_i$, for $1 \leq i \leq n$, InputS operation, the replication dimensions are given by ι_{CD_i};
  - C_{OP}, InputB operation and replication dimensions $\iota_{C_{OP}}$;
  - Arg_j, for $1 \leq j \leq m$, *opinputs* operation, and replication dimensions ι_{Arg_j};
- Regular nodes
  - I, IfDef operation; the replication dimensions are given by α; the control input is C_I, with coordinate mapping $\lambda_{C_I I}$; the node has n pairs of check-data inputs - the i-th pair is $Check_i$ and $Data_i$, with the coordinate mapping $\lambda_{CD_i I}$ and contracted dimensions γ_i;
  - OP, any operation returning bit string value and having one control and $m+1$ data inputs; the replication dimensions are given by β; the control input is C_{OP}, with coordinate mapping $\lambda_{C_{OP}OP}$; the input of the k-th (for a single k, $1 \leq j \leq m$) input port is I, with coordinate mapping λ_{IOP}; for all $j \neq k$, $1 \leq j \leq m$ the input of the j-th input port is either Arg_j with coordinate mapping $\lambda_{Arg_j OP}$ (if $j < k$) or Arg_{j-1} with coordinate mapping $\lambda_{Arg_{j+1}OP}$ (if $j > k$)
- Outputs
  - O_I, OutputS operation, replication dimensions α,the node input is I, with identity coordinate mapping;
  - O_{OP}, OutputS operation, replication dimension β, the node input is OP, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - I, as defined in the initial DGFR;

- A_i, for $1 \leq i \leq n$, And operation, the replication dimensions $\beta + \gamma_i$. The operation inputs are: C_I, with $\lambda_{IOP} \circ \lambda_{C_I I}$ coordinate mapping; $Check_i$, with the $\lambda_{IOP} \circ \lambda_{CD_i I}$ coordinate mapping, and C_{OP}, with $\lambda_{C_{OP}OP}$ coordinate mapping;
- OP'_i, same operation as for OP; the replication dimensions $\beta + \gamma_i$; the control input is A_i, with identity coordinate mapping; the input of the k-th input port is $Data_i$, with identity coordinate mapping; for all $j \neq k$, $1 \leq j \leq m$ the input of the j-th input port is either Arg_j with coordinate mapping $\lambda_{Arg_j OP}$ (if $j < k$) or Arg_{j-1} with coordinate mapping $\lambda_{Arg_{j+1}OP}$ (if $j > k$);
- I', IfDef operation; the replication dimensions β; the control input is $C_O P$, with coordinate mapping $\lambda_{C_{OP}OP}$; the node has n pairs of check-data inputs - the i-th pair is $Check_i$ and OP'_i, with the coordinate mapping $\lambda_{CD_i I}$ and contracted dimensions γ_i;

• Outputs

- O_I, defined as in the initial DGFR;
- O_{OP}, OutputS operation, replication dimension β, the node input is I', with identity coordinate mapping.

The similar transformation can be performed if the OP is an operation returning a boolean value. In this case, the node I' has the operation CfDef, which has the same semantics as IfDef, but operates on boolean values (i.e. all the $Data_i$ are boolean-type inputs). Later on, the CfDef operation can be decomposed into the combination of And, Or, and Longor operations.

Removing the coordinates from IfDef inputs

Let the initial DGFR be the following:

• Inputs

- *Control*, InputB operation, replication dimensions ι_C;
- $Check_i$, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{CD_i};
- $Data_i$, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{CD_i};

• Regular nodes

- I, IfDef operation, let α be its dimensions; let *Control* be its control input with dimension mapping λ_{CI}; let for ($1 \leq i \leq n$) the

$(Check_i, Data_i)$ be its pairs of check- and data-inputs, with dimensions $\beta_c i$ and $\beta_d i$, dimension mappings λ_{C_i} and λ_{D_i}, correspondingly, and contracted dimensions γ_i.

- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I, with identity coordinate mapping.

The transformation is possible if, for some j the $Check_j$ has more coordinates then $Data_j$. Then (given we have such j) , the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - LO, Longor operation, with dimensions $\beta_d j$, input $Check_j$), with dimension mapping λ_{C_j}, and all the coordinates, not present in $\beta_d i$, contracted;
  - I', IfDef operation, with dimensions α; control input $Control$ (with dimension mapping λ_{CI}); let for $(1 \le i \le n,\ i \ne j)$ the $(Check_i, Data_i)$ be its pairs of check- and data-inputs, with dimensions $\beta_c i$ and $\beta_d i$, dimension mappings λ_{C_i} and λ_{D_i}, correspondingly, and contracted dimensions γ_i. The j-th pair of inputs is $(LO, Data_j)$, with dimension mappings $\lambda_{LOI'}$ and λ'_{D_j}, correspondingly, and contracted dimensions γ'_j.
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I', with identity coordinate mapping.

The newly introduced dimensions γ'_j and coordinate mappings $\lambda_{LOI'}$, λ'_{D_j} are identical to γ_j, λ_{C_j}, and λ_{D_j}, except for not having the dimensions, contracted in LO.

Implied Analysis

Introducing $\Rightarrow$ at Boolean Nodes

Let the initial DGFR be the following:

- Inputs
  - S_i, for $1 \le i \le n$, InputB or InputS operation, replication dimensions ι_{S_i};

- Regular nodes
  - B, any operation having n inputs and returning boolean value; let β be its dimensions. The i-th input is S_i, with coordinate mapping $\lambda_{S_i B}$;
- Outputs
  - O_B, OutputB operation, replication dimensions β, and input B, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - B, same as in the initial DGFR
  - Dep_B, $\Rightarrow$, operation, replication dimensions β, the inputs are:
    - B, with identity coordinate mapping;
    - B, with identity coordinate mapping;
- Outputs - same as in the initial DGFR.

Introducing $\Rightarrow$ at Bit String Nodes

Let the initial DGFR be the following:

- Inputs
  - S_i, for $1 \leq i \leq n$, InputB or InputS operation, replication dimensions ι_{S_i};
- Regular nodes
  - S, any operation having n inputs and returning bit string value; let β be its dimensions. The i-th input is S_i, with coordinate mapping $\lambda_{S_i B}$;
- Outputs
  - O_S, OutputS operation, replication dimensions β, and input S, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;

- Regular nodes
  - S, same as in the initial DGFR
  - OK_S, the operation is IsOK, the number of dimensions is β, the input is S, with identity dimension mapping;
  - Dep_S, the operation is $\Rightarrow$, the number of dimensions is β, the inputs are:
    - OK_S, with identity coordinate mapping;
    - OK_S, with identity coordinate mapping;
- Outputs — same as in the initial DGFR.

Introducing $\Rightarrow$ at Or Nodes

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \le i \le n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - O, Or operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_i be the dimension mapping of i-th input $(1 \le i \le n)$.
- Outputs
  - O_O, OutputB operation, replication dimensions α, and input O, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - O, same as in the initial DGFR
  - Dep_i, $1 \le i \le n$, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:
    - B_i, with the dimension mapping $\lambda(d, n).\lambda_i(d, n)$;
- Outputs - same as in the initial DGFR.

Introducing $\Rightarrow$ at Longor Nodes

Let the initial DGFR be the following:

- Inputs
  - B, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - LO, Longor operation; let α be its dimensions and γ the dimensions it contracts; let B be its input; let λ_B be the dimension mapping of the input.
- Outputs
  - O_{LO}, OutputB operation, replication dimensions α, and input LO, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - O, same as in the initial DGFR
  - Dep, the operation is $\Rightarrow$, the number of dimensions is $\alpha + \gamma$. The inputs are:
    - B, with the dimension mapping $\lambda(d, n).\lambda_B(d, n)$;
    - LO, with identity dimension mapping.
- Outputs - same as in the initial DGFR.

Introducing $\Rightarrow$ at And Nodes

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_i be the coordinate mapping of i-th input $(1 \leq i \leq n)$.
- Outputs

  - O_A, $\mathsf{OutputB}$ operation, replication dimensions α, and input A, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - A, same as in the initial DGFR
  - Dep_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:
    - A, with identity dimension mapping;
    - B_i, with the dimension mapping $\lambda(d, n).\lambda_i(d, n)$.
- Outputs - same as in the initial DGFR.

Introducing $\Rightarrow$ at Bit String-to-Boolean Nodes

Let the initial DGFR be the following:

- Inputs
  - S_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{S_i};
- Regular nodes
  - B: IsEq, IsNeq, $\mathsf{TestSig}$, or $\mathsf{TestSigP}$ operation; let α be its dimensions; let $S_1, \ldots, S_n$ be its inputs; let λ_i be the coordinate mapping of i-th input ($1 \leq i \leq n$);
- Outputs
  - O_B, $\mathsf{OutputB}$ operation, replication dimensions α, and input B, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - B, same as in the initial DGFR
  - OK_i, $1 \leq i \leq n$, the operation is IsOK, the number of dimensions is α. The input is S_i, with the dimension mapping $\lambda(d, n).\lambda_i(d, n)$;
  - Dep_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:

    * B, with identity dimension mapping;
    * OK_i, with identity dimension mapping.

- Outputs — same as in the initial DGFR.

Introducing $\Rightarrow$ at Nodes with Control Dependency

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - S_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{S_i};
- Regular nodes
  - S: Nonce, Const, Keypair, PubKey, SigVer, VerKey, SymKey, PubEnc, SymEnc, PubEncZ, SymEncZ, Signature, SignedMsg, Tuple, Proj, PubDec, SymDec, Send, Begin, End, Receive, Secret, Merge, DTakeDimEq, or Id operation; let α be its dimensions; let C be its control input with dimension mapping λ_{CS}; let $S_1, \ldots, S_n$ be the data inputs; let λ_{S_iS} be the dimension mapping of i-th input ($1 \leq i \leq n$);
- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - S, same as in the initial DGFR
  - OK, the operation is IsOK, the number of dimensions is α. The input is S, with identity dimension mapping;
  - Dep_C, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:
    * OK, with identity dimension mapping;
    * C, with the dimension mapping $\lambda(d, n).\lambda_C(d, n)$;
  - OK_i, $1 \leq i \leq n$, the operation is IsOK, the number of dimensions is α. The input is X_i, with the dimension mapping $\lambda(d, n).\lambda_i(d, n)$;

  - Dep_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:
    - S, with identity dimension mapping;
    - OK, with identity dimension mapping.

- Outputs — same as in the initial DGFR.

Introducing $\Rightarrow$ at IfDef-Nodes

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - $Check_i$, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{CD_i};
  - $Data_i$, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{CD_i};
- Regular nodes
  - I, IfDef operation, let α be its dimensions; let C be its control input with coordinate mapping λ_{CI}. The node has n check-data pairs; the i-th pair is $Check_i$, $Data_i$ with coordinate mapping λ_{CD_iI};
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - I, same as in the initial DGFR
  - OK, the operation is IsOK, the number of dimensions is α. The input is I, with identity dimension mapping;
  - Dep_C, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:
    - OK, with identity dimension mapping;
    - C, with the dimension mapping $\lambda(d,n).\lambda_C(d,n)$.
- Outputs — same as in the initial DGFR.

Introducing $\Rightarrow$ at IfDef-Nodes with Single Pair of Inputs

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - *Check*, InputB operation, replication dimensions ι_{CD};
  - *Data*, InputS operation, replication dimensions ι_{CD};
- Regular nodes
  - I, IfDef operation, let α be its dimensions; let the $(\mathit{Check}, \mathit{Data})$ be its only pair of check- and data-inputs, with dimensions β, dimension mapping λ_{CDI}, and contracted dimensions γ;
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - I, same as in the initial DGFR
  - LO_{Check}, operation is Longor, the number of dimensions α, the contracted dimensions γ. The inputs is *Check*, with dimension mapping λ_{CD};
  - OK_D, operation is IsOK, the number of dimensions β. The input is *Data*, with identity dimension mapping;
  - LO_{Data}, operation is Longor, the number of dimensions α, the contracted dimensions γ. The input is OK_D, with dimension mapping λ_{CD};
  - OK_I, operation is IsOK, the number of dimensions α. The input is I, with identity dimension mapping;
  - And_{CD}, operation is And, the number of dimensions α. The inputs are:
    - LO_{Check}, with identity dimension mapping;
    - LO_{Data}, with identity dimension mapping;
  - Dep, the operation is $\Rightarrow$, the number of dimensions is α. The inputs are:

    * OK_I, with identity dimension mapping;
    * And_{CD}, with identity dimension mapping;

- Outputs — same as in the initial DGFR.

Transitive Relation on ⇒ Nodes

Let the initial DGFR be the following:

- Inputs
  - B_1, B_2, B_3, InputB operations, replication dimensions ι_{B_1}, ι_{B_2}, and ι_{B_3}, correspondingly;
- Regular nodes
  - A_1, ⇒ operation; let α_1 be its replication dimensions; let B_1 and B_2 be its inputs, with dimension mappings $\lambda_{B_1A_1}$ and $\lambda_{B_2A_1}$, correspondingly;
  - A_2, ⇒ operation, with α_2 replication dimension, and two inputs: B_2 and B_3, with dimension mappings $\lambda_{B_2A_2}$ and $\lambda_{B_3A_2}$, correspondingly.
- No outputs

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - A_1, A_2, same as in the initial DGFR;
  - A_3, the operation is ⇒, the number of dimensions is α_3. The inputs are:
    * B_1, with dimension mapping $\lambda_{B_1A_3}$;
    * B_3, with dimension mapping $\lambda_{B_3A_3}$;
- Outputs - same as in the initial DGFR.

The coordinate mappings $\lambda_{B_1A_3}$ and $\lambda_{B_3A_3}$ can be defined in any way, as long as the following requirement is met:

$$\begin{aligned}
&\forall i, 1 \leq i \leq k : \\
&\qquad \text{if } \exists j.\lambda_{B_3A_2}(d,j) = \lambda_{B_2A_2}(d, \lambda^{-1}_{B_2A_1}(d, \lambda_{B_1A_1}(d,i))) \text{ then} \\
&\qquad \lambda_{B_1A_3}(d,i) = \lambda_{B_3A_3}(d,j) \\
&\qquad \text{otherwise } \forall j : \lambda_{B_1A_3}(d,i) \neq \lambda_{B_3A_3}(d,j)
\end{aligned}$$

The number of dimensions $\alpha(3)$ is the minimum number of dimensions required to allow defining the dimension mappings this way. Formally, the number of dimensions of the newly created node can be defined as follows:

$$\alpha_3(d) = \Sigma_{i=1}^{k} \begin{cases} \frac{1}{2} & \text{if } \exists j.\lambda_{B_3A_2}(d,j) = \lambda_{B_2A_2}(d, \lambda_{B_2A_1}^{-1}(d, \lambda_{B_1A_1}(d,i))) \\ 1 & \text{if } \forall j.\lambda_{B_3A_2}(d,j) \neq \lambda_{B_2A_2}(d, \lambda_{B_2A_1}^{-1}(d, \lambda_{B_1A_1}(d,i))) \end{cases} +$$

$$+ \Sigma_{j=1}^{k} \begin{cases} \frac{1}{2} & \text{if } \exists i.\lambda_{B_1A_1}(d,i) = \lambda_{B_2A_1}(d, \lambda_{B_2A_2}^{-1}(d, \lambda_{B_3A_2}(d,j))) \\ 1 & \text{if } \forall i.\lambda_{B_1A_1}(d,i) \neq \lambda_{B_2A_1}(d, \lambda_{B_2A_2}^{-1}(d, \lambda_{B_3A_2}(d,j))) \end{cases}$$

Propagating the $\Rightarrow$ through the And-Nodes

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operations, replication dimensions ι_{B_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_{B_iA} be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - D_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is β_i. The inputs are:
    - C, with dimension mapping λ_{CD_i}
    - B_i, with the dimension mapping $\lambda_{B_iD_i}$.
- Outputs
  - O_A, OutputB operation, replication dimensions α, input A, with identity coordinate mapping;

Additionally, we require that the initial DGFR dimension mappings guarantee the (constant) correspondence of coordinate of the And inputs — the following equality should either hold or both parts of it are to be undefined for any i ($1 \leq i \leq n$) and j ($1 \leq j \leq n$):

$$\forall d, \forall l : \lambda_{B_iA}^{-1}(d, \lambda_{B_jA}(d,l)) =$$
$$= \lambda_{B_iD_i}^{-1}(d, \lambda_{CD_i}(d, \lambda_{CD_j}^{-1}(d, \lambda_{B_jD_j}(d,l))))$$

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;

- Regular nodes
  - A, D_i (for $1 \leq i \leq n$), same as in the initial DGFR;
  - D, the operation is $\Rightarrow$, the number of dimensions is γ. The inputs are:
    - C, with dimension mapping λ_{CD}
    - A, with the dimension mapping λ_{AD}.
- Outputs - same as in the initial DGFR.

The dimension mappings λ_{CD} and λ_{AD} can be defined in any way, as long as the following requirement is met:

$$\forall i, 1 \leq i \leq n, \forall d, \forall l : \\ \lambda^{-1}_{B_i D_i}(d, \lambda_{CD_i}(d, l)) = \lambda^{-1}_{B_i A}(d, \lambda^{-1}_{AD}(d, \lambda_{CD}(d, l)))$$

The number of dimensions γ is the minimum number of dimensions required to allow defining the λ_{CD} and λ_{AD} this way.

Propagating the $\Rightarrow$ through the Or-Nodes

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operations, replication dimensions ι_{B_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - O, Or operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let $\lambda_{B_i O}$ be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - D_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is β_i. The inputs are:
    - B_i, with the dimension mapping $\lambda_{B_i D_i}$.
    - C, with dimension mapping λ_{CD_i}
- Outputs
  - O_O, OutputB operation, replication dimensions α, input O, with identity coordinate mapping;

The requirements we put on the dimension mappings are the same as in case of the previous transformation: the following should hold for any i $(1 \leq i \leq n)$ and j $(1 \leq j \leq n)$:

$$\begin{aligned}\forall d, \forall l : \lambda^{-1}_{B_iO}(d, \lambda_{B_jO}(d, l) = \\ = \lambda^{-1}_{B_iD_i}(d, \lambda_{CD_i}(d, \lambda^{-1}_{CD_j}(d, \lambda_{B_jD_j}(d, l))))\end{aligned}$$

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - O, D_i (for $1 \leq i \leq n$), same as in the initial DGFR;
  - D, the operation is $\Rightarrow$, the number of dimensions is γ. The inputs are:
    - O, with the dimension mapping λ_{OD}.
    - C, with dimension mapping λ_{CD}
- Outputs — same as in the initial DGFR.

The dimension mappings λ_{CD} and λ_{OD} can be defined in any way, as long as the following requirement is met:

$$\begin{aligned}\forall i, 1 \leq i \leq n, \forall d, \forall l : \\ \lambda^{-1}_{B_iD_i}(d, \lambda_{CD_i}(d, l)) = \lambda^{-1}_{B_iO}(d, \lambda^{-1}_{OD}(d, \lambda_{CD}(d, l)))\end{aligned}$$

The number of dimensions γ is the minimum number of dimensions required to allow defining the λ_{CD} and λ_{OD} this way.

Transitive Relation on $\Rightarrow$ Nodes through the Longor Nodes

Let the initial DGFR be the following:

- Inputs
  - B_1, B_2, B_3, InputB operations, replication dimensions ι_{B_1}, ι_{B_2}, and ι_{B_3}, correspondingly;
- Regular nodes
  - LO_{21}, Longor operation; let β_{21} be its dimensions and γ_{21} the dimensions it contracts; its input is B_2, with dimension mapping $\lambda_{B_2LO_{21}}$;

  - LO_{22}, Longor operation; let β_{22} be its dimensions and γ_{22} the dimensions it contracts; its input is B_2, with dimension mapping $\lambda_{B_2 LO_{22}}$.
  - A_1, $\Rightarrow$ operation; let α_1 be its replication dimensions; let B_1 and LO_{21} be its inputs, with dimension mappings $\lambda_{B_1 A_1}$ and $\lambda_{LO_{21} A_1}$, correspondingly;
  - A_2, $\Rightarrow$ operation, with α_2 replication dimensions, and two inputs: LO_{22} and B_3, with dimension mappings $\lambda_{LO_{22} A_2}$ and $\lambda_{B_3 A_2}$, correspondingly.

- Outputs
  - $O_{LO_{21}}$, OutputB operation, replication dimensions β_{21}, input LO_{21} with identity coordinate mapping;
  - $O_{LO_{22}}$, OutputB operation, replication dimensions β_{22}, input LO_{22} with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - LO_{21}, LO_{22}, A_1, and A_2, as in the initial DGFR;
  - LO_{13}, Longor operation; let β_{13} be its dimensions and γ_{13} the dimensions it contracts; its input is B_1, with dimension mapping $\lambda_{B_1 LO_{13}}$;
  - LO_{33}, Longor operation; let β_{33} be its dimensions and γ_{33} the dimensions it contracts; its input is B_3, with dimension mapping $\lambda_{B_3 LO_{33}}$;
  - A_3, the operation is $\Rightarrow$, the number of dimensions is α_3. The inputs are:
    * LO_{13}, with dimension mapping $\lambda_{LO_{13} A_3}$;
    * LO_{33}, with dimension mapping $\lambda_{LO_{33} A_3}$.
- Outputs - same as in the initial DGFR.

The dimensions contracted by the newly introduced Longor operations are defined as follows - every dimension of B_1, corresponding to a dimension, contracted in LO_{22}, should be contracted in LO_{13}. Similarly, every dimension of B_2, corresponding to a dimension, contracted in LO_{21}, should be contracted in LO_{33}.

The dimension mappings $\lambda_{B_1 LO_{13}}$, $\lambda_{B_3 LO_{33}}$, $\lambda_{LO_{13} A_3}$, and $\lambda_{LO_{33} A_3}$ can be chosen freely, as long as two dimensions of B_1 and B_3 correspond to the same dimension of A_3 if and only if they correspond to the same dimension of B_2. The number of dimensions of the newly introduced nodes is the minimum number of dimensions required to allow defining the dimension mappings this way.

Propagating the $\Rightarrow$ through a sequence of And and Longor Nodes

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operations, replication dimensions ι_{B_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let $\lambda_{B_i A}$ be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - LOB_i, $1 \leq i \leq n$, the operation is Longor; let β_{LOB_i} be its dimensions and γ_{LOB_i} the dimensions it contracts; its input is B_i, with dimension mapping $\lambda_{B_i LOB_i}$;
  - LOC_i, $1 \leq i \leq n$, the operation is Longor; let β_{LOC_i} be its dimensions and γ_{LOC_i} the dimensions it contracts; its input is C, with dimension mapping $\lambda_{C LOC_i}$;
  - D_i, $1 \leq i \leq n$, the operation is $\Rightarrow$, the number of dimensions is δ_i. The inputs are:
    * LOC_i, with dimension mapping $\lambda_{LOC_i D_i}$
    * LOB_i, with the dimension mapping $\lambda_{LOB_i D_i}$.
- Outputs
  - O_A, OutputB operation, replication dimensions α, input A, with identity coordinate mapping;
  - O_{LOB_i}, $1 \leq i \leq n$, OutputB operation, replication dimensions β_{LOB_i}, and the input LOB_i, with identity coordinate mapping;
  - O_{LOC_i}, $1 \leq i \leq n$, OutputB operation, replication dimensions β_{LOC_i}, and the input LOC_i, with identity coordinate mapping.

Similarly to the transformation without Longor, we require that the dimension mappings present in the initial DGFR guarantee the (constant) correspondence of coordinate of the And inputs — i.e. that irrespective of whether we "trace" the coordinate of B_i to a coordinate of B_j through the A node, or through the LOB_i, D_i, LOC_i, C, LOC_j, D_j, and LOB_j, the same coordinate correspondence relations should be obtained. If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - A, LOB_i, LOC_i, and D_i (for $1 \leq i \leq n$), same as in the initial DGFR;
  - LO_C, Longor operation; let β_{LO_C} be its dimensions and γ_{LO_C} the dimensions it contracts; its input is C, with dimension mapping λ_{CLO_C};
  - LO_A, Longor operation; let β_{LO_A} be its dimensions and γ_{LO_A} the dimensions it contracts; its input is A, with dimension mapping λ_{ALO_A};
  - D, the operation is $\Rightarrow$, the number of dimensions is δ. The inputs are:
    - LO_C, with dimension mapping λ_{LO_CD}
    - LO_A, with the dimension mapping λ_{LO_AD}.
- Outputs - same as in the initial DGFR.

The dimensions contracted by the newly introduced Longor operations are defined as follows — every dimension of C, corresponding to a dimension, contracted in every LOB_i ($1 \leq i \leq n$), can be contracted in LO_C. Similarly, every dimension of A, corresponding to a dimension, contracted in LOC_i, shall be contracted in LO_A.

The dimension mappings λ_{CLO_C}, λ_{ALO_A}, λ_{LO_CD}, and λ_{LO_AD} can be chosen freely, as long as two dimensions of A and C correspond to the same dimension of D if and only if they correspond to the same dimension of each B_i. The numbers of dimensions β_{LO_C}, β_{LO_A}, and δ is the minimum number of dimensions required to allow defining the dimension mappings this way.

Propagating the $\Rightarrow$ through a sequence of Or and Longor Nodes

The transformation is identical to the previous one, except for the direction of implied dependencies (the order of the arguments to the $\Rightarrow$ operations is the opposite).

The $\Rightarrow$ removal

Every transformation, defined above, has a dual transformation, defined as following: the new DGFR (result of the transformation) can be replaced with the original DGFR, without affecting the DGFR semantics. Note that it is not allowed to just remove the $\Rightarrow$ operations, as, in the absence of the rest of the DGFR-result of the transformation, they can turn to $\top$, thus stopping the execution of the DGF, corresponding to the DGFR.

Introducing Merge

The transformation described in this section use the result of the implied analysis. If the control dependency of some operation implies one if its input being equal to some other value, this input is replaced with the Merge (the inputs are the values being equal).

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - X_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{X_i};
  - X, InputS operation, replication dimensions ι_X;
- Regular nodes
  - S: Nonce, Const, Keypair, PubKey, SigVer, VerKey, SymKey, PubEnc, SymEnc, PubEncZ, SymEncZ, Signature, SignedMsg, Tuple, Proj, PubDec, SymDec, Send, Begin, End, Receive, Secret, Merge, DTakeDimEq, or Id operation; let β be its dimensions; let C be its control input with dimension mapping λ_{CS}; let $X_1, \ldots, X_n$ be the data inputs; let λ_{X_iS} be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - A, IsEq operation; let α be its dimensions; let X_j (for some j, such that $1 \leq j \leq n$) and X be its inputs, with dimension mappings λ_{X_jA} and λ_{XA}, correspondingly.
  - D, the operation is $\Rightarrow$, the number of dimensions is γ. The inputs are:
    - C, with dimension mapping λ_{CD}
    - A, with dimension mapping λ_{AD}.
- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping;

We require that the dimension mappings present in the initial DGFR guarantees the correspondence of coordinate of the X inputs, independent of chosen edge path — i.e. that irrespective of whether we "trace" the coordinate of X_j to a coordinate of S directly (using the λ_{X_jA} dimension mapping), or through the A, D, and C nodes, the same coordinate correspondence relations should be obtained.

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - M, Merge operation, with dimensions α. The control input is A, with identity dimensions mapping. The data inputs are X_j and X, with dimension mappings λ_{X_jA} and λ_{XA}, correspondingly.
  - S': same operation as S; dimensions are β; C is its control input with dimension mapping λ_{CS}; The data inputs are $X_1, \ldots, X_{j-1}, M, X_{j+1}, \ldots, X_n$; the dimension mapping of i-th input $(1 \leq i \leq n, i \neq j)$ is λ_{X_iS}. The dimension mapping of M is $\lambda_{MS'}$.
- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S', with identity coordinate mapping;

The newly introduced dimension mapping $\lambda_{MS'}$ can be chosen freely, as long as two dimensions of X_j and X correspond to the same dimension of S' if and only if they correspond to the same dimension of D.

Combining two Merge

This transformation use the result of the implied analysis. If the result of one Merge is used as an input to another, then the inputs of the first operation could be used as the inputs of the second.

Let the initial DGFR be the following:

- Inputs
  - B_1, InputB operation, replication dimensions ι_{B_1};
  - B_2, InputB operation, replication dimensions ι_{B_2};
  - X_{1i}, for $1 \leq i \leq n$, InputS operation, replication dimensions $\iota_{X_{1i}}$;

 - X_{2j}, for $1 \leq i \leq m-1$, InputS operation, replication dimensions $\iota_{X_{2j}}$;

- Regular nodes
  - M_1, with dimensions α, control input B_1 with dimension mapping $\lambda_{B_1M_1}$, and data inputs $X_{11}, \ldots, X_{1n}$. The dimension mapping of i-th data input $(1 \leq i \leq n)$ is $\lambda_{X_{1i}M_1}$.
  - M_2, with dimensions β, control input B_2 with dimension mapping $\lambda_{B_2M_2}$, and data inputs $X_{21}, \ldots, X_{2j-1}, M_1, X_{2j+1}, \ldots, X_{2m}$. The dimension mapping of i-th input $(1 \leq i \leq m, i \neq j)$ is $\lambda_{X_{2i}M_1}$. The dimension mapping of j-th input is $\lambda_{M_1M_2}$.

- Outputs
  - O_{M_1}, OutputS operation, replication dimensions α, and input M_1, with identity coordinate mapping;
  - O_{M_2}, OutputS operation, replication dimensions β, and input M_2, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - M_1, as in the initial DGFR;
  - M_3, Merge operation, with dimensions β. The control input is B_2, with dimensions mapping $\lambda_{B_2M_2}$. The $(m+n-1)$ data inputs are:
    * For each i, $1 \leq i \leq n$: X_{1i}, with dimension mapping $\lambda(d,n) = \lambda_{M_1M_2}(d, \lambda_{X_{1i}M_1}(d,n))$;
    * For each i, $1 \leq i \leq m, i \neq j$: X_{2i}, with dimension mapping $\lambda(d,n) = \lambda_{X_{2i}M_2}(d,n)$.
- Outputs
  - O_{M_1}, as in the initial DGFR;
  - O_{M_2}, OutputS operation, replication dimensions β, and input M_3, with identity coordinate mapping;

Removing And inputs, depending on each other

If an input of the the And operation implies another input of the same operation, that another input can be removed from the inputs list.

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let $\lambda_{B_i A}$ be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - D, the operation is $\Rightarrow$, the number of dimensions is β. The inputs are:
    - B_{i_1}, with dimension mapping $\lambda_{B_{i_1} D}$
    - B_{i_2}, with dimension mapping $\lambda_{B_{i_2} D}$.
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A, with identity coordinate mapping;

We require that the dimension mappings present in the initial DGFR guarantee the (constant) correspondence of coordinate of the And inputs — i.e. that irrespective of whether we "trace" the coordinate of B_{i_1} to a coordinate of B_{i_2} through the A node, or through the D node, the same coordinate correspondence relations should be obtained.

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - D, as on the initial DGFR;
  - A', And operation with dimensions α, inputs $B_1, \ldots, B_{i_2-1}, B_{i_2+1}, \ldots, B_n$. For i-th input the dimension mapping is $\lambda_{B_i A}$;
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A', with identity coordinate mapping;

Removing Or inputs, depending on each other

If an input of the the Or operation implies another input of the same operation, the first input can be removed from the inputs list.

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - A, Or operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_{B_iA} be the dimension mapping of i-th input $(1 \leq i \leq n)$.
  - D, the operation is $\Rightarrow$, the number of dimensions is β. The inputs are:
    - B_{i_1}, with dimension mapping $\lambda_{B_{i_1}D}$
    - B_{i_2}, with dimension mapping $\lambda_{B_{i_2}D}$.
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A, with identity coordinate mapping;

We require that the dimension mappings present in the initial DGFR guarantee the (constant) correspondence of coordinate of the Or inputs — i.e. that irrespective of whether we “trace” the coordinate of B_{i_1} to a coordinate of B_{i_2} through the A node, or through the D node, the same coordinate correspondence relations should be obtained.

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - D, as on the initial DGFR;
  - A', Or operation with dimensions α, inputs $B_1, \ldots, B_{i_1-1}, B_{i_1+1}, \ldots, B_n$. For i-th input the dimension mapping is λ_{B_iA}.
- Outputs
  - O_A, OutputB operation, replication dimensions α, and input A', with identity coordinate mapping;

Removing IfDef inputs not affecting the computation result

Let the initial DGFR be the following:

- Inputs
  - *Control*, InputB operation, replication dimensions ι_C;
  - *Check*$_i$, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{CD_i};
  - *Data*$_i$, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{CD_i};
- Regular nodes
  - I, IfDef operation, let α be its dimensions; let *Control* be its control input with dimension mapping λ_{CI}; let for $(1 \leq i \leq n)$ the $(\mathit{Check}_i, \mathit{Data}_i)$ be its pairs of check- and data-inputs, with dimensions β_i, dimension mapping λ_{CD_i}, and contracted dimensions γ_i.
  - LO, Longor operation with dimensions δ, input *Check*$_j$ (for some j, $1 \leq j \leq n$), having dimension mapping $\lambda_{\mathit{Check}_j O}$ and contracted dimensions δ_c;
  - *Dep*, $\Rightarrow$ operation with dimensions ϵ, the following inputs:
    * *Control*, dimension mapping $\lambda_{\mathit{ControlDep}}$;
    * LO, dimension mapping λ_{LODep};
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I, with identity coordinate mapping.

Additionally we require that all the coordinates contracted in j-th pair of inputs (i.e. in γ_j), are also contracted in the LO operation.

If the requirements are met, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - LO, *Dep*, as on the initial DGFR;
  - I', IfDef operation, with dimensions α, control input *Control* (with dimension mapping λ_{CI}); and a single pair of inputs $(\mathit{Check}_j, \mathit{Data}_j)$ with dimension mapping λ_{CD_j}, and contracted dimensions γ_j.
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I', with identity coordinate mapping.

Adding Coordinates to the Longor nodes

Let the initial DGFR be the following:

- Inputs
  - B, InputB operation, replication dimensions ι_B;
  - B_i, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{B_i};
- Regular nodes
  - LO_1, Longor operation with dimensions β, input B with dimension mapping λ_{BLO_1} and contracted dimensions β_c;
  - A, And operation with dimensions γ, $n+1$ inputs $LO_1, B_1, \ldots, B_n$, having dimension mappings $\lambda_{LO_1A}, \lambda_{B_1A}, \ldots, \lambda_{B_nA}$
  - LO_2, Longor operation with dimensions δ, input B with dimension mapping λ_{BLO_2} and contracted dimensions δ_c;
  - *Dep*, $\Rightarrow$ operation with dimensions ϵ. The operation has the following inputs:
    * A, with dimension mapping λ_{AD};
    * LO_2, with dimension mapping λ_{LO_2D};
- Outputs
  - O_{LO_1}, OutputB operation, replication dimensions β, input LO_1 with identity coordinate mapping;
  - O_{LO_2}, OutputB operation, replication dimensions δ, input LO_2 with identity coordinate mapping;
  - O_A, OutputB operation, replication dimensions γ, input A with identity coordinate mapping;

If a coordinate of B, having the corresponding coordinate in A (through the sequence of nodes LO_2 and *Dep*), is contacted in LO_1, the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - LO_1, LO_2, same as in the initial DGFR;
  - LO'_1, Longor operation with dimensions β', input B with coordinate mapping $\lambda_{BLO'_1}$ and contracted dimensions β'_c;
  - A', And operation, replication dimensions γ, $n+1$ inputs $LO'_1, B_1, \ldots, B_n$, with coordinate mappings $\lambda_{LO'_1A}, \lambda_{B_1A}, \ldots, \lambda_{B_nA}$;

- Outputs
  - O_{LO_1}, O_{LO_2}, same as in the initial DGFR;
  - O_A, OutputB operation, replication dimensions γ, input A' with identity coordinate mapping.

The newly introduced dimension mappings, contracted dimensions, and node dimensions are chosen to be equal to the corresponding mappings of the initial DGFR, except for the coordinates of B, having the corresponding coordinate in A (through the sequence of nodes LO_2 and Dep), are mapped to the same coordinate of A through the LO'_1.

Introducing the CTakeDimEq at Longor

Let the initial DGFR be the following:

- Inputs
  - B, InputB operation, replication dimensions α;
- Regular nodes
  - LO, Longor operation, with dimensions β and contracted dimensions β_c; the input to the operation is B, with coordinate mapping λBLO;
  - DE, DimEq operation with dimensions γ, and the dimension d coordinates i and j compared;
  - Dep, with dimensions δ, and the following inputs:
    * B, with coordinate mapping λ_{BDep};
    * DE, with coordinate mapping λ_{DEDep};
- Outputs
  - O_{LO}, OutputB operation, replication dimensions β, and input LO, with identity coordinate mapping;
  - O_{DE}, OutputB operation, replication dimensions γ, and input DE, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - DE, Dep, as in the initial DGFR;

  - $CTDE$, CTakeDimEq operation, with dimensions ϵ, the dimension d coordinates i' and j' compared; the input to the operation is B, with coordinate mapping λ_{BCTDE};
  - LO', Longor operation, with dimensions β and contracted dimensions β_c; the input to the operation is B, with coordinate mapping λBLO;

- Outputs
  - O_{DE}, as in the initial DGFR;
  - $O_{LO'}$, OutputB operation, replication dimensions β, and input LO', with identity coordinate mapping;

The newly introduced dimensions and the coordinate mappings should map the dimension d coordinates i and j of the DE node to the coordinates i' and j' of the $CTDE$ node, and the rest of the coordinates in the same way as λBLO does.

Introducing the CTakeDimEq and DTakeDimEq at IfDef

Let the initial DGFR be the following:

- Inputs
  - $Control$, InputB operation, replication dimensions ι_C;
  - $Check_i$, for $1 \le i \le n$, InputB operation, replication dimensions ι_{CD_i};
  - $Data_i$, for $1 \le i \le n$, InputS operation, replication dimensions ι_{CD_i};
- Regular nodes
  - I, IfDef operation, let β be its dimensions; let $Control$ be its control input with dimension mapping λ_{CI}; let for $(1 \le i \le n)$ the $(Check_i, Data_i)$ be its pairs of check- and data-inputs, with dimensions β_i, dimension mapping λ_{CD_i}, and contracted dimensions γ_i.
  - DE, DimEq operation with dimensions γ, and the dimension d coordinates c_i and c_j compared;
  - Dep, with dimensions δ, and the following inputs:
    - $Check_j$ (for some j, $1 \le j \le n$), with coordinate mapping $\lambda_{Check_j Dep}$;
    - DE, with coordinate mapping λ_{DEDep};
- Outputs

  - O_I, OutputS operation, replication dimensions β, and input I, with identity coordinate mapping.
  - O_{DE}, OutputB operation, replication dimensions γ, and input DE, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - DE, Dep, as in the initial DGFR;
  - $CTDE$, CTakeDimEq operation, with dimensions ϵ_c, the dimension d coordinates c'_i and c'_j compared; the node input is $Check_j$, with coordinates mapping $\lambda Check_j CTDE$;
  - $DTDE$, CTakeDimEq operation, with dimensions ϵ_d, the dimension d coordinates c'_i and c'_j compared; the node input is $Data_j$, with coordinates mapping $\lambda Data_j DTDE$; the control input is the control input of the $Data_j$, with the corresponding coordinates mapping;
  - I', IfDef operation with dimensions β and control input $Control$ with dimension mapping λ_{CI}; The data inputs are:
    - For $(1 \leq i \leq n,\ i \neq j)$ the $(Check_i, Data_i)$, with coordinates mapping λ_{CD_i}, and contracted dimensions γ_i.
    - The $(CTDE, DTDE)$, with coordinates mapping λ'_{CD_j}, and contracted dimensions γ'_j.
- Outputs
  - O_{DE}, as in the initial DGFR;
  - O_I, OutputS operation, replication dimensions β, and input I', with identity coordinate mapping.

The newly introduced dimensions and the coordinate mappings should map the dimension d coordinates i and j of the DE node to the coordinates i' and j' of the $CTDE$ and $DTDE$ nodes, and the rest of the coordinates in the same way mappings on the initial DGFR do.

Replacing the CTakeDimEq or DTakeDimEq with its Input

Let the initial DGFR be the following:

- Inputs
  - X, InputB or InputS operation, with dimensions α;

- Regular nodes
  - D, CTakeDimEq and DTakeDimEq operation with dimensions β, and dimension d and coordinate i compared (with the coordinate $\beta(d) + 1$). The operation input is X, with coordinate mapping λ_{XD}. If the operation of the node is DTakeDimEq, then let C be its control inputs, with dimension mapping λ_{CD}
- Outputs
  - O_D, OutputS or OutputS operation, replication dimensions β, and input D, with identity coordinate mapping.

If one or both dimension d coordinates i, $\beta(d) + 1$, are not present (through the λ^{-1}_{XD} mapping) in X node, then the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- No regular nodes
- Outputs
  - O_D, OutputS or OutputS operation, replication dimensions β, and input X, with identity coordinate mapping.

Propagating the CTakeDimEq and DTakeDimEq to the Node Inputs

Let the initial DGFR be the following:

- Inputs
  - X_i, for $1 \leq i \leq n$, InputB or InputS operation, with dimensions ι_{X_i};
- Regular nodes
  - X, any operation with n inputs and replication dimensions α; the i-th input is X_i, with the coordinate mapping $\lambda_{X_i X}$;
  - D, CTakeDimEq and DTakeDimEq operation with dimensions β, and dimension d and coordinate i compared (with the coordinate $\beta(d) + 1$). The operation input is X, with coordinate mapping λ_{XD}. If the operation of the node is DTakeDimEq, then let C be its control inputs, with dimension mapping λ_{CD}
- Outputs

  - O_D, OutputS or OutputS operation, replication dimensions β, and input D, with identity coordinate mapping.

Additionally we require that both dimension d coordinates i, $\beta(d)+1$, are present (through the λ^{-1}_{XD} mapping) in X node.

If the requirements are met, then the initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - For $1 \leq i \leq n$, if both dimension d coordinates i, $\beta(d)+1$, are present (through the λ^{-1}_{XD} and $\lambda^{-1}_{X_iX}$ mappings) in X_i node, the node X'_i:
    - If X_i returns bit string, then the DTakeDimEq operation (dimension d and coordinates corresponding to i, $\beta(d)+1$ coordinates of D), with input X_i, and control input C;
    - If X_i has the operation returning boolean value, then the CTakeDimEq operation (dimension d and coordinates corresponding to i, $\beta(d)+1$ coordinates of D), with input X_i;
  - D', same operation as X, with dimensions β, and n inputs. The i-th input is:
    - if both dimension d coordinates i, $\beta(d)+1$, are present (through the λ^{-1}_{XD} and $\lambda^{-1}_{X_iX}$ mappings) in X_i node, the node X'_i, with dimension mapping $\lambda_{X_iD'}$;
    - otherwise, the node X_i, with dimension mapping $\lambda_{X'_iD'}$.
- Outputs
  - O_D, OutputS or OutputS operation, replication dimensions β, and input D', with identity coordinate mapping.

The requirement towards the newly introduced dimensions and coordinate mappings is that the coordinates of the X_i nodes should map to the same coordinate of D' as in D.

Not-And Analysis

IsEq and IsNeq operations with same inputs

Let the initial DGFR contain the following nodes:

- Inputs

  - X, InputS operation, replication dimensions ι_X;
  - Y, InputS operation, replication dimensions ι_Y;

- Regular Nodes
  - E, IsEq operation, with dimensions α and inputs X, Y, with dimension mappings λ_{XE} and λ_{YE}, correspondingly;
  - N, IsNeq operation, with dimensions β and inputs X, Y, with dimension mappings λ_{XN} and λ_{YN}, correspondingly;
- Outputs
  - O_E, OutputB operation, replication dimensions α, input E with identity coordinate mapping;
  - O_N, OutputB operation, replication dimensions β, input N with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - E, N, as in the initial DGFR;
  - AMO, Nand operation, with dimensions γ, and the following inputs:
    - E, with dimension mappings λ_{EAMO};
    - N, with dimension mappings λ_{NAMO};
- Outputs — as in the initial DGFR;

The coordinate mappings λ_{EAMO} and λ_{NAMO} can be chosen freely, as long as two coordinates of X and Y correspond to the same coordinate of AMO if and only if they correspond to the same coordinates of E and N. The numbers of coordinates γ is the minimum number of coordinates required to allow defining the coordinate mappings this way.

False **and Any Node Returning Bit String**

Let the initial DGFR contain the following nodes:

- Inputs
  - X, InputS operation, replication dimensions ι_X;
- Regular Nodes

  - F, False operation, no replication dimensions;

- Outputs
  - O_F, OutputB operation, no replication dimensions, input F with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - F, as in the initial DGFR;
  - OK, IsOK operation, with dimensions ι_X, and input X, with identity coordinate mapping;
  - AMO, Nand operation, with dimensions ι_X, and the following inputs:
    * F, with identity dimension mapping;
    * OK, with identity dimension mapping.
- Outputs — as in the initial DGFR;

False and Any Node Returning Boolean Value

Let the initial DGFR contain the following nodes:

- Inputs
  - B, InputB operation, replication dimensions ι_B;
- Regular Nodes
  - F, False operation, no replication dimensions;
- Outputs
  - O_F, OutputB operation, no replication dimensions, input F with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - F, as in the initial DGFR;

  - AMO, Nand operation, with dimensions ι_B, and the following inputs:
    - F, with identity dimension mapping;
    - B, with identity dimension mapping.
- Outputs — as in the initial DGFR;

Error **and Any Node Returning Bit String**

Let the initial DGFR contain the following nodes:

- Inputs
  - X, InputS operation, replication dimensions ι_X;
- Regular Nodes
  - E, Error operation, no replication dimensions;
- Outputs
  - O_E, OutputS operation, no replication dimensions, input E with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - E, as in the initial DGFR;
  - OK_1, IsOK operation, no replication dimensions, and input E, with identity coordinate mapping;
  - OK_2, IsOK operation, replication dimensions ι_X, and input X, with identity coordinate mapping;
  - AMO, Nand operation, with dimensions α, and the following inputs:
    - OK_1, with identity coordinate mapping;
    - OK_2, with identity coordinate mapping.
- Outputs — as in the initial DGFR;

Error **and Any Node Returning Boolean Value**

Let the initial DGFR contain the following nodes:

- Inputs
  - B, InputB operation, replication dimensions ι_B;
- Regular Nodes
  - E, Error operation, no replication dimensions;
- Outputs
  - O_E, OutputS operation, no replication dimensions, input E with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - E, as in the initial DGFR;
  - OK, IsOK operation, no replication dimensions, and input E, with identity dimension mapping;
  - AMO, Nand operation, with dimensions α, and the following inputs:
    - OK_1, with identity coordinate mapping;
    - B, with identity coordinate mapping;
- Outputs — as in the initial DGFR;

Propagating Nand **Through** And

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \le i \le n$, InputB operations, replication dimensions ι_{B_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - A, And operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_i be the dimension mapping of i-th input ($1 \le i \le n$).
  - AMO, Nand operation, with dimensions β, and the following inputs:

    * B_j (for some $j, 1 \le j \le n$), with dimension mapping λ_{B_jAMO};
    * C, with dimension mapping λ_{CAMO}.

- Outputs
  - O_A, OutputB operation, replication dimensions α, input A, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - A, AMO, same as in the initial DGFR;
  - AMO_2, Nand operation, with dimensions α, and the following inputs:
    * A, with identity dimension mapping;
    * C, with dimension mapping $\lambda(d, n).\lambda_i(d, n)$.
- Outputs — same as in the initial DGFR.

Propagating Nand Through the Control Input of Bit String Node

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - X_i, for $1 \le i \le n$, InputS operation, replication dimensions ι_{X_i};
  - D, InputB operation, replication dimensions ι_D;
- Regular nodes
  - S: IfDef, Nonce, Const, Keypair, PubKey, SigVer, VerKey, SymKey, PubEnc, SymEnc, PubEncZ, SymEncZ, Signature, SignedMsg, Tuple, Proj, PubDec, SymDec, Send, Begin, End, Receive, Secret, Merge, DTakeDimEq, or Id operation; let α be its dimensions; let C be its control input with dimension mapping λ_C; let $X_1, \ldots, X_n$ be the data inputs; let λ_i be the dimension mapping of i-th input ($1 \le i \le n$).
  - AMO, Nand operation, with dimensions β, and the following inputs:
    * C, with dimension mapping λ_{CAMO};
    * D, with dimension mapping λ_{DAMO};

- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - S, AMO, same as in the initial DGFR
  - OK, IsOK operation, replication dimensions α and input S, with identity coordinate mapping;
  - AMO_2, Nand operation, replication dimensions α, and the following inputs:
    - OK, with identity dimension mapping;
    - C, with dimension mapping $\lambda(d, n).\lambda_C$;
- Outputs — same as in the initial DGFR.

Propagating Nand Through the Data Input of a Bit String Node

Let the initial DGFR be the following:

- Inputs
  - C, InputB operation, replication dimensions ι_C;
  - X_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{X_i};
  - D, InputB operation, replication dimensions ι_D;
- Regular nodes
  - S: Nonce, Const, Keypair, PubKey, SigVer, VerKey, SymKey, PubEnc, SymEnc, PubEncZ, SymEncZ, Signature, SignedMsg, Tuple, Proj, PubDec, SymDec, Send, Begin, End, Receive, Secret, Merge, DTakeDimEq, or Id operation; let α be its dimensions; let C be its control input with dimension mapping λ_C; let $X_1, \ldots, X_n$ be the data inputs; let λ_i be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - OK_1, IsOK operation, with dimensions γ and input X_j, with identity dimension mapping $\lambda_{X_j OK_1}$;
  - AMO, Nand operation, with dimensions β, and the following inputs:

    - OK_1, with dimension mapping $\lambda_{OK_1 AMO}$;
    - D, with dimension mapping λ_{DAMO};

- Outputs
  - O_S, OutputS operation, replication dimensions α, and input S, with identity coordinate mapping.
  - O_{OK_1}, OutputB operation, replication dimensions γ, and input OK_1, with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - S, OK_1, AMO, same as in the initial DGFR
  - OK, IsOK operation, replication dimensions α and input S, with identity coordinate mapping;
  - AMO_2, Nand operation, replication dimensions α, and the following inputs:
    - OK, with identity dimension mapping;
    - C, with dimension mapping $\lambda(d, n).\lambda_C$.
- Outputs — same as in the initial DGFR.

Propagating Nand Through Or Node

Let the initial DGFR be the following:

- Inputs
  - B_i, for $1 \leq i \leq n$, InputB operations, replication dimensions ι_{B_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - O, Or operation; let α be its dimensions; let $B_1, \ldots, B_n$ be its inputs; let λ_i be the dimension mapping of i-th input ($1 \leq i \leq n$).
  - AMO, Nand operation, with dimensions β, and the following inputs:
    - O, with dimension mapping λ_{OAMO};
    - C, with dimension mapping λ_{CAMO}.
- Outputs

  - O_O, OutputB operation, replication dimensions α, input O, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - O, AMO, same as in the initial DGFR;
  - AMO_i, $1 \leq i \leq n$, the operation is Nand, replication dimensions β. The inputs are:
    - B_i, with the coordinate mapping $\lambda(d, n).\lambda_{OAMO}(d, \lambda_i(d, n))$;
    - C, with coordinate mapping λ_{CAMO}.
- Outputs — same as in the initial DGFR.

Propagating Nand through Longor

Let the initial DGFR be the following:

- Inputs
  - B, InputB operations, replication dimensions ι_B;
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - LO, Longor operation; let α be its dimensions and γ the dimensions it contracts; let B be its input; let λ_B be the coordinate mapping of the input.
  - AMO, Nand operation, with dimensions β, and the following inputs:
    - LO, with coordinate mapping λ_{LOAMO};
    - C, with coordinate mapping λ_{CAMO};
- Outputs
  - O_{LO}, OutputB operation, replication dimensions α, input LO, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes

  - LO, AMO, same as in the initial DGFR;
  - AMO_2, Nand operation, with dimensions δ, and the following inputs:
    - B, with dimension mapping λ_{BAMO_2};
    - C, with dimension mapping λ_{CAMO_2};

- Outputs — same as in the initial DGFR.

The coordinate mappings λ_{BAMO_2} and λ_{CAMO_2} can be chosen freely, as long as two coordinate of B and C correspond to the same coordinate of AMO_2 if and only if they correspond to the same coordinate of AMO. The numbers of coordinate δ is the minimum number of coordinate required to allow defining the coordinate mappings this way.

Propagating Nand through the IfDef

Let the initial DGFR be the following:

- Inputs
  - *Control*, InputB operation, replication dimensions $\iota_{Control}$;
  - *Check*$_i$, for $1 \leq i \leq n$, InputB operation, replication dimensions ι_{CD_i};
  - *Data*$_i$, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{CD_i};
  - C, InputB operation, replication dimensions ι_C;
- Regular nodes
  - I, IfDef operation, let α be its dimensions; let *Control* be its control input with coordinate mapping λ_{CI}; let for $(1 \leq i \leq n)$ the (*Check*$_i$, *Data*$_i$) be its pairs of check- and data-inputs, with dimensions β_i, coordinate mapping λ_{CD_i}, and contracted dimensions γ_i;
  - AMO, Nand operation, with dimensions δ, and the following inputs:
    - I, with coordinate mapping λ_{IAMO};
    - C, with coordinate mapping λ_{CAMO};
- Outputs
  - O_I, OutputS operation, replication dimensions α, and input I, with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;

- Regular nodes
  - I, AMO, same as in the initial DGFR
  - AMO_{Check_i} $(1 \leq i \leq n)$, Nand operation, relication dimensions ϵ, and the following inputs:
    - $Check_i$, with coordinate mapping $\lambda_{CD_iAMO_i}$;
    - C, with coordinate mapping λ_{CAMO_i};
  - OK_{D_i} $(1 \leq i \leq n)$, operation is IsOK, replication dimensions β_i. The input is $Data_i$, with identity coordinate mapping;
  - AMO_{Data_i} $(1 \leq i \leq n)$, Nand operation, replication dimensions ϵ, and the following inputs:
    - OK_{D_i}, with coordinate mapping $\lambda_{CD_iAMO_i}$;
    - C, with coordinate mapping λ_{CAMO_i};
  - $AMO_{Control}$, Nand operation, replication dimensions δ, and the following inputs:
    - $Control$, with coordinate mapping $\lambda(d,n).\lambda_{IAMO}(d,\lambda_{CI}(d,n))$;
    - C, with coordinate mapping λ_{CAMO};
- Outputs — same as in the initial DGFR.

The coordinate mappings $\lambda_{CD_iAMO_i}$ and λ_{CAMO_i} can be chosen freely, as long as two coordinates of $Check_i$ and C correspond to the same coordinate of AMO_{Check_i} (the same also holds for AMO_{Data_i}) if and only if they correspond to the same coordinates of AMO. The replication dimensions ϵ is the minimum number of replication dimensions required to allow defining the coordinate mappings this way.

The Nand operation removal

Every transformation, defined above, has a dual transformation, defined as following: the new DGFR (result of the transformation) can be replaced with the original DGFR, without affecting the DGFR semantics. Note that it is not allowed to just remove the Nand operations, as, in the absence of the rest of the DGFR-result of the transformation, they can turn to $\top$, thus stopping the execution of the DGF, corresponding to the DGFR.

Using the NAND Analysis Results at Boolean Nodes

Let the initial DGFR contain the following nodes:

- Inputs

  - S_i, for $1 \leq i \leq n$, InputB or InputS operation, replication dimensions ι_{S_i};
- Regular Nodes
  - B, any operation having n inputs and returning boolean value; let α be its replication dimensions. The i-th input is S_i, with coordinate mapping λ_{S_iB};
  - AMO, Nand operation, replication dimensions β. The node has the following inputs:
    * B, with coordinate mapping λ_{BAMO};
    * B, with coordinate mapping λ_{BAMO};
- Outputs
  - O_B, OutputB operation, replication dimensions α, input B with identity coordinate mapping;

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - F, False operation, no replication dimensions;
- Outputs
  - O_B, OutputB operation, replication dimensions α, input F with identity coordinate mapping.

Using NAND Analysis Results at Bit String Nodes

Let the initial DGFR contain the following nodes:

- Inputs
  - S_i, for $1 \leq i \leq n$, InputB or InputS operation, replication dimensions ι_{S_i};
- Regular Nodes
  - X, any operation having n inputs and returning bit string value; let α be its replication dimensions. The i-th input is S_i, with coordinate mapping λ_{S_iX};

  - OK, IsOK operation, replication dimensions β and input X, with the coordinate mapping λ_{XOK}
  - AMO, Nand operation, replication dimensions γ. The operation has the following inputs:
    * OK, with coordinate mapping λ_{OKAMO};
    * OK, with coordinate mapping λ_{OKAMO};

- Outputs
  - O_X, OutputS operation, replication dimensions α, input X with identity coordinate mapping;
  - O_{OK}, OutputB operation, replication dimensions β, input OK with identity coordinate mapping.

The initial DGFR can be replaced with the following DGFR:

- Inputs — same as in the initial DGFR;
- Regular nodes
  - F, False operation, no replication dimensions;
  - E, Error operation, no replication dimensions;
- Outputs
  - O_X, OutputS operation, replication dimensions α, input E with identity coordinate mapping;
  - O_{OK}, OutputB operation, replication dimensions β, input F with identity coordinate mapping.

Removing Or Inputs Having No Effect on the Computation Result

Let the initial DGFR be the following:

- Inputs
  - B_{1i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{B_{1i}}$;
  - B_{2j}, for $1 \leq i \leq m$, InputB operation, replication dimensions $\iota_{B_{2j}}$;
  - B, InputB operation, replication dimensions ι_B;
- Regular nodes
  - O, Or operation with dimensions α and n inputs $B_{11}, \ldots, B_{1n}$, having dimension mappings $\lambda_{B_{11}O}, \ldots, \lambda_{B_{1n}O}$;

  - A, And operation with dimensions β and $m+1$ inputs $O, B_{21}, \ldots, B_{2m}$, having dimension mappings $\lambda_{OA}, \lambda_{B_{21}A}, \ldots, \lambda_{B_{2m}A}$;
  - Dep, $\Rightarrow$ operation with dimensions γ, the following inputs:
    * A, dimension mapping λ_{ADep};
    * B, dimension mapping λ_{BDep};
  - Amo, Nand operation with dimensions δ, the following inputs:
    * B, dimension mapping λ_{BAmo};
    * B_{1j} (for some j, $1 \leq j \leq n$), dimension mapping $\lambda_{B_{1j}Amo}$;

- Outputs
  - O_O, OutputB operation, replication dimensions α, and input O, with identity coordinate mapping;
  - O_A, OutputB operation, replication dimensions β, and input A, with identity coordinate mapping.

Additionally we require that the dimension mappings present in the initial DGFR guarantee that the coordinates of B_{1j} correspond to the same dimensions of Amo, irrespective whether we "trace" the coordinate of B_{1j} through the nodes O, A, Dep, and B; or directly through the mapping the $\lambda_{B_{1j}Amo}$.

If the requirements are met, the initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - Amo, O, as in the initial DGFR;
  - O', Or operation with dimensions α and $n-1$ inputs $B_{11}, \ldots, B_{1j-1}, B_{1j+1}, \ldots, B_{1n}$, with coordinate mappings $\lambda_{B_{11}O}, \ldots, \lambda_{B_{1n}O}$;
  - A', And operation with dimensions β and $m+1$ inputs $O', B_{21}, \ldots, B_{2m}$, with coordinate mappings $\lambda_{OA}, \lambda_{B_{21}A}, \ldots, \lambda_{B_{2m}A}$;
  - Dep, $\Rightarrow$ operation with dimensions γ, the following inputs:
    * A', dimension mapping $\lambda_{A'Dep}$;
    * B, dimension mapping λ_{BDep};
- Outputs
  - O_O, as in the initial DGFR.
  - O_A, OutputB operation, replication dimensions β, and input A', with identity coordinate mapping.

Removing Longor Inputs Having no Effect on the Computation Result

Let the initial DGFR be the following:

- Inputs
  - B_1, InputB operation, replication dimensions ι_{B_1};
  - B_{2j}, for $1 \leq i \leq m$, InputB operation, replication dimensions $\iota_{B_{2j}}$;
  - B, InputB operation, replication dimensions ι_B;
- Regular nodes
  - LO, Longor operation, replication dimensions α, input B_1, with coordinate mapping λ_{B_1O} and contracted dimensions α_c;
  - A, And operation, replication dimensions β and $m+1$ inputs $LO, B_{21}, \ldots, B_{2m}$, with coordinate mappings $\lambda_{LOA}, \lambda_{B_{21}A}, \ldots, \lambda_{B_{2m}A}$;
  - Dep, $\Rightarrow$ operation, replication dimensions γ, and the following inputs:
    - A, coordinate mapping λ_{ADep};
    - B, coordinate mapping λ_{BDep};
  - Amo, Nand operation, replication dimensions δ, and the following inputs:
    - B, coordinate mapping λ_{BAmo};
    - B_1, coordinate mapping λ_{B_1Amo};
- Outputs
  - O_{LO}, OutputB operation, replication dimensions α, and input LO, with identity coordinate mapping;
  - O_A, OutputB operation, replication dimensions β, and input A, with identity coordinate mapping.

Additionally we require that at least one B_1 coordinate, contracted in LO, has corresponding (through the nodes Amo, B, and Dep) A coordinate. The rest of the coordinates of B_1 should correspond to the same dimensions of Amo, irrespective whether we "trace" the coordinate of B_1 through the nodes LO, A, Dep, and B; or directly through the mapping the λ_{B_1Amo}.

If the requirements are met, the initial DGFR can be replaced with the following:

- Inputs - same as in the initial DGFR;

- Regular nodes
  - Amo, LO, as in the initial DGFR;
  - LO', Longor operation, replication dimensions α', input B_1, with coordinate mapping λ_{B_1O} and contracted dimensions α'_c;
  - A', And operation, replication dimensions β and $m+1$ inputs $O', B_{21}, \ldots, B_{2m}$, with coordinate mappings $\lambda_{OA}, \lambda_{B_{21}A}, \ldots, \lambda_{B_{2m}A}$;
  - Dep, $\Rightarrow$ operation with dimensions γ, the following inputs:
    - A', dimension mapping $\lambda_{A'Dep}$;
    - B, dimension mapping λ_{BDep};
- Outputs
  - O_{LO}, as in the initial DGFR;
  - O_A, OutputB operation, replication dimensions β, and input A', with identity coordinate mapping;

The newly introduced dimensions and dimension mappings should map the B_1 coordinates, previously contracted in LO, but having the corresponding A coordinate (through the Amo, B, and Dep mappings), to that A coordinate.

Cryptographic Primitives

Symmetric Encryption Replacement

We require the encryption system to satisfy the IND-CCA and cipher text integrity properties defined in Appendix 1. The encryption of the plain text with the given public key is replaced with the encryption of the string of zeroes (or any other constant) of equal length, and that latter cipher text is indistinguishable from the first for anyone, except for the one having the corresponding key. On the decryption side we first check whether the cipher text matches one of the cipher texts already produced, and if it does, the corresponding plain text is returned. The difference with the asymmetric encryption is that if the cipher text is not equal to one of the cipher texts produced, the result of the decryption is, due to the cipher text integrity, always $\perp$.

Formally, let the initial DGFR be the following:

- Inputs
  - C_{RK}, InputB operation, replication dimensions ι_{RK};
  - C_K, InputB operation, replication dimensions ι_K;
  - C_{RE_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{RE_i}}$;

  - C_{E_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{E_i}}$;
  - C_{D_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions $\iota_{C_{D_j}}$;
  - PT_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{PT_i};
  - CT_j, for $1 \leq j \leq m$, InputS operation, replication dimensions ι_{CT_j};

- Regular nodes
  - RK, RS operation, the replication dimensions α, the control input C_{RK}, with coordinate mapping $\lambda_{C_{RK}RK}$;
  - K, SymKey operation, the replication dimensions α, the control input C_K, with coordinate mapping $\lambda_{C_K K}$; the data input RK with identity coordinate mapping;
  - RE_i, for $1 \leq i \leq n$, RS operation, the replication dimension β_i, the control input C_{RE_i}, with coordinate mapping $\lambda_{C_{RE_i}RE_i}$;
  - E_i, for $1 \leq i \leq n$, SymEnc operation, the replication dimension β_i, the control input C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$; the data inputs are:
    * Random coins: RE_i, with identity coordinate mapping;
    * Key: K, with coordinate mapping λ_{KE_i};
    * Plain text: PT_i, with coordinate mapping $\lambda_{PT_iE_i}$;
  - D_j, for $1 \leq j \leq m$, SymDec operation, the replication dimension γ_j, the control input C_{D_j}, with coordinate mapping $\lambda_{C_{D_j}D_j}$; the data inputs are:
    * Key: K, with coordinate mapping λ_{KD_j};
    * Cypher text: CT_j, with coordinate mapping $\lambda_{CT_jD_j}$;

- Outputs
  - O_{E_i}, for $1 \leq i \leq n$, the $\mathsf{OutputS}$ operation, the replication dimensions β_i, the input E_i, with identity coordinate mapping;
  - O_{D_j}, for $1 \leq j \leq m$, the $\mathsf{OutputS}$ operation, the replication dimensions γ_j, the input D_j, with identity coordinate mapping;

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - RK, RS operation, the replication dimensions α, the control input C_{RK}, with coordinate mapping $\lambda_{C_{RK}RK}$;

  - K, SymKey operation, the replication dimensions α, the control input C_K, with coordinate mapping $\lambda_{C_K K}$; the data input RK with identity coordinate mapping;
  - RE_i, for $1 \leq i \leq n$, RS operation, the replication dimensions β_i, the control input C_{RE_i}, with coordinate mapping $\lambda_{C_{RE_i} RE_i}$;
  - OK_i, for $1 \leq i \leq n$, IsOK operation, the replication dimensions β_i, the input is PT_i, with coordinate mapping $\lambda_{PT_i E_i}$;
  - A_i, for $1 \leq i \leq n$, And operation, the replication dimensions β_i, the operation has two inputs: C_{E_i}, with coordinate mapping $\lambda_{C_{E_i} E_i}$, and OK_i, with identity coordinate mapping;
  - E'_i, for $1 \leq i \leq n$, SymEncZ operation, the replication dimensions β_i, the control input A_i, with identity coordinate mapping; the data inputs are:
    * Random coins: RE_i, with identity coordinate mapping;
    * Key: K, with coordinate mapping λ_{KE_i};
  - EQ_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, IsEq operation, the replication dimension $\gamma_j + \beta_i$ (pointwise addition), the inputs are:
    * CT_j, with identity coordinate mapping;
    * E'_i, with coordinate mapping $\lambda(d, n).\gamma_j(d) + n$;
  - D'_j, $1 \leq j \leq m$, IfDef operation, the replication dimension γ_j, the control input C_{D_j}, with coordinate mapping $\lambda_{C_{D_j} D_j}$; the operation has n pair of data inputs: for $1 \leq i \leq n$, the check-input is EQ_{ij}, with the identity coordinate mapping; the data-input is PT_i, with coordinate mapping $\lambda(d, n).\gamma_j(d) + \lambda_{PT_i E_i}$; the contracted dimensions are β_i

- Outputs
  - O_{E_i}, for $1 \leq i \leq n$, the OutputS operation, the replication dimensions β, the input E'_i, with identity coordinate mapping;
  - O_{D_j}, for $1 \leq j \leq m$, the OutputS operation, the replication dimensions γ, the input D'_j, with identity coordinate mapping;

Theorem 8. For the symmetric encryption transformation described above, the public view of the transformed DGF H' (corresponding to the transformed DGFR) is indistinguishable from the public view initial DGF H (corresponding to the initial DGFR).

Proof. In the fragment H' the $\rho[O_{E_i}]$ returns the encryption of the string of zeroes of the length equal to the length of the plain text $\rho[PT_i]$ and the

$\rho[O_{D_j}] = \rho[D'_j]$ is one of the plain texts earlier encrypted by the fragment, or $\perp$ (if the cipher text being decrypted has not been encrypted by the fragment).

It is possible that $\rho[D'_j]$ would be equal to $\top$. The IfDef operation can only return $\top$ if more than one node EQ_{ij} returns true; the probability that it happens is negligible, as the values compared with (the result of the node E'_i) are functions of independent random coins (generated at the nodes with labels RE_i).

Having shown that $\rho[D'_j]$ (and thus $\rho[O_{D_j}]$) can be equal to $\top$ only with negligible probability, we go on with demonstrating that from the driver algorithm $\mathcal{A}_H$, producing the public views of H from H' and the algorithm A, distinguishing these public views, the algorithm $\mathcal{A}_E$, winning the IND-CCA game with the same probability, can be constructed.

The $\mathcal{A}_E$ behaves like $\mathcal{A}_H$, but instead of returning to the graph fragment evaluation function, it calls the symmetric encryption oracle defined in Appendix 1:

- when $\mathcal{A}_H$ first sets the $\rho[C_K]$ and $\rho[C_{RK}]$ to true the $\mathcal{A}_E$ executes $\underline{\text{Initialize}}$;
- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv}$) the $\rho[C_{E_i}.\overline{cv}]$ and $\rho[C_{RE_i}.\overline{cv}]$ to true, and the $\rho[PT_i.\overline{cv}]$ to M_0, the $\mathcal{A}_E$ executes $\underline{\text{LR}}(M_0, (0)^{|M_0|})$, and puts the returned value to $\rho[O_{E_i}.\overline{cv}]$,
- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv'}$) the $\rho[C_{D_j}.\overline{cv'}]$ to true and the $\rho[CT_j.\overline{cv'}]$ to C, the $\mathcal{A}_E$ executes $\underline{\text{Dec}}(C)$, and puts the returned value to $\rho[O_{D_j}.\overline{cv'}]$,
- finally, when $\mathcal{A}_H$ indicates to stop with the public view, and the algorithm A is executed to produce the bit b, indicating whether the public view corresponds to H or H', the $\mathcal{A}_E$ calls $\underline{\text{Finalize}}(b)$.

By examining the semantics of the graph fragment step functions, it can be checked that indeed the evaluation of H or H' will set the outputs according to the rules implemented by $\mathcal{A}_E$ above. As all the random coins, including those used by encryption oracle and by the RS operation on the graphs, are generated according to the same distribution, so the keys generated in node K and encryption oracle in the game are.

So, the $\mathcal{A}_E$ will win the IND-CCA game with the same probability as the A will distinguish the view $\mathsf{view}^H(\mathcal{A}_H)$ from $\mathsf{view}^{H'}(\mathcal{A}_H)$. By the requirement we put on the encryption scheme, this probability is negligible. $\square$

Signature Replacement

We require the signature scheme abstracted by the operations SigVer, Verkey, Signature, SignedMsg, and TestSig to be ACMA-secure, as defined in Appendix 1. The idea of the transformation is: if the secret key used to produce the signatures is used for this purpose only (and, therefore, is unknown to the driver algorithm), the signature verification operation is replaced with a conjunction of the signature verification operation and a check that the message being verified belongs to the set of messages previously signed with the same key.

Formally, let the initial DGFR be the following:

- Inputs
  - C_{RK}, InputB operation, replication dimensions ι_{RK};
  - C_{SV}, InputB operation, replication dimensions ι_{SV};
  - C_{VK}, InputB operation, replication dimensions ι_{VK};
  - C_{RS_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{RS_i}}$;
  - C_{S_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{S_i}}$;
  - M_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{M_i};
  - C_{SM_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions $\iota_{C_{SM_j}}$;
  - $C_{SM_jS_k}$, for $1 \leq j \leq m$ and $1 \leq k \leq k_j$, InputS operation, replication dimensions $\iota_{SM_jS_k}$
- Regular nodes
  - RK, RS operation, the replication dimensions α, the control input C_{RK}, with coordinate mapping $\lambda_{C_{RK}RK}$;
  - SV, SigVer operation, the replication dimensions α, the control input C_{SV}, with coordinate mapping $\lambda_{C_{SV}SV}$; the data input RK with identity coordinate mapping;
  - VK, VerKey operation, the replication dimensions α, the control input C_{VK}, with coordinate mapping $\lambda_{C_{VK}VK}$; the data input SV with identity coordinate mapping;
  - RS_i, for $1 \leq i \leq n$, RS operation, the replication dimension β_i, the control input C_{RS_i}, with coordinate mapping $\lambda_{C_{RS_i}RS_i}$;
  - S_i, for $1 \leq i \leq n$, $\mathsf{Signature}$ operation, the replication dimension β_i, the control input C_{S_i}, with coordinate mapping $\lambda_{C_{S_i}S_i}$; the data inputs are:
    - ∗ Random coins: RS_i, with identity coordinate mapping;
    - ∗ Key: SV, with coordinate mapping λ_{SVS_i};
    - ∗ Message: M_i, with coordinate mapping $\lambda_{M_iS_i}$;

  - SM_j, for $1 \leq j \leq m$, any operation returning a bit string value; the replication dimensions are ι_{SM_j}. The control input is C_{SM_j} (with coordinate mapping $\lambda_{C_{SM_j}SM_j}$). The data inputs are (for $1 \leq k \leq k_j$) SM_jS_k, with coordinate mapping $\lambda_{SM_jS_k}SM_j$;
  - V_j, for $1 \leq j \leq m$, TestSig operation, the replication dimension γ_j, with the following inputs:
    * Key: VK, with coordinate mapping λ_{VKV_j};
    * Signed message: SM_j, with coordinate mapping $\lambda_{SM_jV_j}$;

- Outputs
  - O_{VK}, the OutputS operation, the replication dimensions α, the input VK, with identity coordinate mapping;
  - O_{S_i}, for $1 \leq i \leq n$, the OutputS operation, the replication dimensions β_i, the input S_i, with identity coordinate mapping;
  - O_{SM_j}, for $1 \leq j \leq m$, the OutputS operation, the replication dimensions ι_{SM_j}, the input SM_j, with identity coordinate mapping;
  - O_{V_j}, for $1 \leq j \leq m$, the OutputB operation, the replication dimensions γ_j, the input V_j, with identity coordinate mapping.

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;
- Regular nodes
  - RK, SV, VK, RS_i ($1 \leq i \leq n$), S_i ($1 \leq i \leq n$), and V_j ($1 \leq j \leq m$) nodes, defined as in the initial DGFR;
  - SM'_j, for $1 \leq j \leq m$, SignedMsg operation, replication dimensions ι_{SM_j}, and the following inputs:
    * Control: C_{SM_j}, with coordinate mapping $\lambda_{C_{SM_j}SM_j}$;
    * Message: SM_j, with identity coordinate mapping;
  - EQ_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, IsEq operation, replication dimensions $\iota_{SM_j} + \beta_i$ (pointwise addition), the inputs are:
    * SM'_j, with identity coordinate mapping;
    * M_i, with coordinate mapping $\lambda(d, n).\iota_{SM_j}(d) + n$;
  - A_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, And operation, replication dimensions $\iota_{SM_j} + \beta_i$ (pointwise addition). The operation has two inputs: C_{S_i}, with coordinate mapping $\lambda_{C_{S_i}S_i}$, and EQ_{ij}, with identity coordinate mapping;

  - LO_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, Longor operation, replication dimensions γ_j, and all the coordinates, present in the $\iota_{SM_j} + \beta_i$, but not present in γ_j, contracted;
  - O_j, for $1 \leq j \leq m$, Or operation, replication dimensions γ_j, and n inputs. The i-th input ($1 \leq i \leq n$) is LO_{ij}, with identity coordinate mapping;
  - A_j, for $1 \leq j \leq m$, And operation, replication dimensions γ_j. The operation has two inputs: O_j and V_j, both with identity coordinate mapping;

- Outputs
  - O_{VK}, O_{S_i}, and O_{SM_j} nodes, defined as in the initial DGFR;
  - O_{V_j}, for $1 \leq j \leq m$, the OutputB operation, the replication dimensions γ_j, the input A_j, with identity coordinate mapping.

Theorem 9. For the digital signature transformation described above, the public view of the transformed DGF H' (corresponding to the transformed DGFR) is indistinguishable from the public view initial DGF H (corresponding to the initial DGFR).
Proof. The only difference between the H and H' is how the $\rho[O_{V_j}]$ is computed – in H it is a result of TestSig operation, while in H' it is a concatenation of the TestSigP operation (having the same semantics as TestSig; we use different names just to avoid the repeated application of this transformation) and checking that the message being verified belongs to the set of the messages previously signed at S_i (for all i, $1 \leq i \leq n$).

In order to prove the theorem, we demonstrate that from the driver algorithm $\mathcal{A}_H$, producing the public views of H from H' and the algorithm A, distinguishing these public views, the algorithm $\mathcal{A}_S$, winning the ACMA game (as defined in Appendix 1) with the same probability, can be constructed.

The $\mathcal{A}_S$ behaves like $\mathcal{A}_H$, but instead of returning to the graph fragment evaluation function, it calls the oracle defined in Appendix 1:

- When $\mathcal{A}_H$ first sets the $\rho[C_{RK}]$, $\rho[C_{SV}]$ to true the $\mathcal{A}_S$ executes $\underline{\text{Initialize}}$ and saves the returned public key;
- when $\mathcal{A}_H$ first sets the $\rho[C_{VK}]$ to true the $\mathcal{A}_S$ sets the $\rho[O_{VK}]$ to the saved public key;
- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv}$) the $\rho[C_{RS_i}\overline{cv}]$ and $\rho[C_{S_i}.\overline{cv}]$ to true, and the $\rho[M_i.\overline{cv}]$ to M, the $\mathcal{A}_S$ executes $\underline{\text{Sign}}(\rho[M.\overline{cv}])$, and puts the message M, concatenated with the signature (returned value) to $\rho[O_{S_i}.\overline{cv}]$;

- when $\mathcal{A}_H$ first sets (for some coordinate vector $\overline{cv'}$) the $\rho[C_{SM_j}.\overline{cv'}]$ to true and (for the given j and all k, $1 \leq k \leq k_j$) the $\rho[C_{SM_jS_k}.\overline{cv'}]$ to non-$\perp$ values, the $\mathcal{A}_S$ first computes the $\rho[SM_j.\overline{cv'}]$ and sets the $\rho[O_{SM_j}.\overline{cv'}]$ to the resulting value. Then, it executes $\underline{\mathsf{Verify}}(V,\ O_{SM_j}.\overline{cv'})$, and puts the returned value to $\rho[O_{V_j}].\overline{cv'}$;
- finally, when $\mathcal{A}_H$ indicates to stop with some public view, $\mathcal{A}_S$ also stops.

By examining the semantics of the graph fragment step functions, it can be checked that indeed the evaluation of H or H' will set the outputs according to the rules implemented by $\mathcal{A}_S$ above. As all the random coins, including those used by encryption oracle and by the RS operation, are generated according to the same distribution, so the keys generated at SV and signature oracle in the game are.

Since the algorithm A is able to distinguish the public views, and the $\rho[O_{VK}]$, $\rho[O_{S_i}]$, and $\rho[O_{SM_j}]$ are computed according to the same formulae, the values of $\rho[O_{V_j}]$ should be different. Given the semantics of the graph, it is only possible that for some j and coordinate vector $\overline{cv'}$ the $\rho[O_{V_j}].\overline{cv'}$ is true in H, but is false in H' — it means that some message successfully passed the signature test operation, but it has not been signed by the signature oracle (all the checks A_j fail), and, therefore, the ACMA game is won. By the requirement we put on the signature scheme, this probability is negligible. □

Public Key Analysis

In this transformation, we consider the value used as a public key in asymmetric encryption operation, and analyze separately the cases where it is equal to each of the public keys corresponding to the key pairs generated in the protocol, and the case where it is not equal to any of those public keys.

Formally, let the initial DGFR be the following:

- Inputs
  - C_{RK_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions ι_{RK_j};
  - C_{KP_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions ι_{KP_j};
  - C_{PK_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions ι_{PK_j};
  - C_{D_j}, for $1 \leq j \leq m$, InputB operation, replication dimensions $\iota_{C_{D_j}}$;
  - CT_j, for $1 \leq j \leq m$, InputS operation, replication dimensions ι_{CT_j};
  - C_{E_i}, for $1 \leq i \leq n$, InputB operation, replication dimensions $\iota_{C_{E_i}}$;

  - PK'_i, for $1 \leq i \leq n$, InputS operation, replication dimensions $\iota_{PK'_i}$;
  - R_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{R_i};
  - PT_i, for $1 \leq i \leq n$, InputS operation, replication dimensions ι_{PT_i};

- Regular nodes
  - RK_j, RS operation, the replication dimensions α, the control input C_{RK_j}, with coordinate mapping $\lambda_{C_{RK_j}RK_j}$;
  - KP_j, Keypair operation, the replication dimensions α, the control input C_{KP_j}, with coordinate mapping $\lambda_{C_{KP_j}KP_j}$; the data input RK_j with identity coordinate mapping;
  - PK_j, PubKey operation, the replication dimensions α, the control input C_{PK_j}, with coordinate mapping $\lambda_{C_{PK_j}PK_j}$; the data input KP_j with identity coordinate mapping;
  - D_j, for $1 \leq j \leq m$, PubDec operation, the replication dimension γ_j, the inputs are:
    - Control: C_{D_j}, with coordinate mapping $\lambda_{C_{D_j}D_j}$;
    - Key: KP_j, with coordinate mapping $\lambda_{PK_jD_j}$;
    - Cypher text: CT_j, with coordinate mapping $\lambda_{CT_jD_j}$;
  - E_i, for $1 \leq i \leq n$, PubEnc operation, the replication dimension β_i, the inputs are:
    - Control C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$;
    - Random coins: RE_i, with identity coordinate mapping;
    - Key: PK'_i, with coordinate mapping $\lambda_{PK'_iE_i}$;
    - Plain text: PT_i, with coordinate mapping $\lambda_{PT_iE_i}$;
- Outputs
  - O_{PK_j}, the OutputS operation, the replication dimensions α, the input PK_j, with identity coordinate mapping;
  - O_{E_i}, for $1 \leq i \leq n$, the OutputS operation, the replication dimensions β_i, the input E_i, with identity coordinate mapping;
  - O_{D_j}, for $1 \leq j \leq m$, the OutputS operation, the replication dimensions γ_j, the input D_j, with identity coordinate mapping;

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;
- Regular nodes

- RK_j, KP_j, PK_j, and D_j nodes, defined as in the initial DGFR;
- EQ_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, IsEq operation, the replication dimensions $\beta_i + \gamma_j$ (pointwise addition), the inputs are:
  * PK'_i, with coordinate mapping $\lambda(d,n).\lambda_{PK'_iE_i}(d,n)$;
  * PK_j, with coordinate mapping $\beta_i(d) + \lambda_{PK_jD_j}$;
- NEQ_{ij}, for $1 \leq i \leq n$ and $1 \leq j \leq m$, IsNeq operation, the replication dimensions $\beta_i + \gamma_j$ (pointwise addition), the inputs are:
  * PK'_i, with coordinate mapping $\lambda(d,n).\lambda_{PK'_iE_i}(d,n)$;
  * PK_j, with coordinate mapping $\beta_i(d) + \lambda_{PK_jD_j}$;
- A_i, for $1 \leq i \leq n$, And operation. The replication dimensions are $\beta_i + \gamma_j$ (pointwise addition). The operation has m inputs, the j-th input is NEQ_{ij}1
- PK''_i, $1 \leq i \leq n$, IfDef operation, the replication dimension β_i, the control input C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$; the operation has $m+1$ pair of data inputs:
  * for $1 \leq j \leq m$, the check-input is EQ_{ij}, with the identity coordinate mapping; the data-input is PK_j, with coordinate mapping $\beta_i(d) + \lambda_{PK_jD_j}$; the contracted dimensions are γ_j;
  * for $i+1$-th pair, the check-input is A_i, with identity coordinate mapping; the data-input is PK'_i, with coordinate mapping $\lambda(d,n).\lambda_{PK'_iE_i}(d,n)$; the contracted dimensions are γ_j;
- E'_i, for $1 \leq i \leq n$, PubEnc operation, the replication dimension β_i, the inputs are:
  * Control C_{E_i}, with coordinate mapping $\lambda_{C_{E_i}E_i}$;
  * Random coins: RE_i, with identity coordinate mapping;
  * Key: PK''_i, with identity coordinate mapping;
  * Plain text: PT_i, with coordinate mapping $\lambda_{PT_iE_i}$;

- Outputs
  - O_{PK_j} and O_{D_j}, defined as in the initial DGFR;
  - O_{E_i}, for $1 \leq i \leq n$, the OutputS operation, the replication dimensions β_i, the input E'_i, with identity coordinate mapping;

The correctness of this transformation follows from the definition of the semantics of the IfDef operation, and the structure of the transformed DGFR — it is obvious that (with non-negligible probability) any given value can be equal to at most one public keys considered (as the public keys are extracted from the public-secret key pairs, each generated from random coins (used only for that), and are not used for anything else than for the extraction of the public key component or the decryption (using the secret key component). Therefore, for any given coordinate vector $\overline{cv}$, exactly one of the check-connections of PK''_i will always be set to true (so, IfDef will not return $\top$ or $\perp$ if its control dependency (the same as for the E_i) is true), and, as the data-connections will always return the same public key used in the original operation, the E'_i will use the same key as the corresponding E_i operation in the initial DGFR. So, the $\rho[O_{E_i}]$ are computed in the initial and the transformed DGFR in the same way, as the remaining outputs are.

Decryption of the result of $\mathsf{PubEncZ}$

Formally, let the initial DGFR be the following:

- Inputs
  - R, InputS operation, replication dimensions ι_R;
  - K, InputS operation, replication dimensions ι_K;
  - X, InputS operation, replication dimensions ι_X;
  - Y, InputS operation, replication dimensions ι_Y;
- Regular nodes
  - P, $\mathsf{PubEncZ}$ operation; let α be its dimensions; let R and K be its randomness, and public key inputs, with coordinate mappings λ_{RP} and λ_{KP}, correspondingly;
  - E, IsEq operation, with dimensions β and inputs P, X, with dimension mappings λ_{PE} and λ_{XE}, correspondingly;
  - D, PubDec operation, with dimensions γ, and inputs X, Y, with dimension mappings λ_{XD} and λ_{YD}, correspondingly;
- Outputs
  - O_P, $\mathsf{OutputS}$ operation, replication dimensions α, input P with identity coordinate mapping;
  - O_E, $\mathsf{OutputB}$ operation, replication dimensions β, input E with identity coordinate mapping;
  - O_D, $\mathsf{OutputS}$ operation, replication dimensions γ, input D with identity coordinate mapping;

The DGFR can be replaced with the following one:

- Inputs - same as in the initial DGFR;

- Regular nodes
  - OK, IsOK operation, with dimensions γ, and input D, with identity dimension mapping;
  - AMO, Nand operation, with dimensions δ, and the following inputs:
    - OK, with dimension mappings λ_{OKAMO};
    - E, with dimension mappings λ_{EAMO}.
- Outputs - same as in the initial DGFR;

The dimension mappings λ_{OKAMO} and λ_{DAMO} can be chosen freely, as long as two dimensions of X correspond to the same dimension of AMO regardless of whether the mapping is taken through the E node, or through the sequence of D and OK nodes. The numbers of dimensions δ is the minimum number of dimensions required to allow defining the dimension mappings this way.