Computational Soundness

- The Case of Diffie-Hellman Keys -

Emmanuel BRESSON [a] Yassine LAKHNECH [b] Laurent MAZARÉ [c] and Bogdan WARINSCHI [d]

[a] *DCSSI Crypto Lab, Paris, France*
[b] *VERIMAG Grenoble, Grenoble, France*
[c] *LexiFI, Paris, France*
[d] *University of Bristol, United Kingdom*

1. Introduction

1.1. Background

Symbolic vs. Computational Models. A common criticism of symbolic approaches for security is that they rely on models that are too abstract to offer clear security guarantees. In such models the adversary appears to be severely restricted. The axioms that characterize security of primitives allow the adversary only a limited number of operations, and it is usually unclear how to enforce these axioms in actual implementations. Furthermore, the representation of messages as symbols does not permit reasoning about partial information, a real concern in many applications.

This criticism is even more relevant given alternative models that offer clearly stronger guarantees. Under these *computational* models, security analysis of protocols considers a much lower level of abstraction. Typically, parties are viewed as algorithms (written in some Turing-complete language) and the messages with which they operate and communicate are actual bitstrings. The adversaries are required to operate efficiently (*i.e.* run in time polynomial in some security parameter), but are otherwise allowed to perform *arbitrary* computations. Furthermore, unlike in the case of symbolic methods where security of primitives is axiomatized, in computational approaches security is defined. This enables rigorous proofs that implementations actually meet their required security levels starting from widely accepted assumptions.

The technique most commonly used in such proofs is known as "reduction". The idea is to show that the security of a cryptographic construct can be reduced to solving some problem(s) that is believed to be hard. Specifically, one argues that any adversary that is successful against the cryptographic construct can be used to solve some underlying hard problem. Typical hard problems include factoring, taking discrete logarithms, the computational Diffie-Hellman problem, etc [24].

The low level of abstraction and the powerful and realistic adversarial model imply strong security guarantees schemes with computational security proofs. Unfortunately, reduction techniques do not scale well. While they enjoyed considerable success for

primitives and small protocols, more complex cryptographic constructs are notoriously difficult to analyze rigorously. Nowadays it is quite common that proofs are either high level sketches that are missing important details, or they are detailed but in this case they are virtually impossible to check. For symbolic methods, the situation is in some sense reversed. As discussed above, the high level of abstraction, the axiomatic way of assuming security of primitives, and the rather limited adversary imply that the symbolic security proofs do not have clear implications for actual implementations. However, proofs can be, and usually are, quite rigorous. More importantly, symbolic analysis may benefit from machine support or may even be completely automated, as explained in previous chapters.

1.2. Symbolic techniques and computational analysis

Motivated in part by the unsettling gap between the two co-existing approaches, and in part by the realization that computational proofs could benefit from the rigourousness specific to symbolic methods, a significant research effort attempts to investigate the links between symbolic and computational methods for security analysis. The goal is to tame the complexity specific to computational models via symbolic techniques. Existing research falls along two main directions.

Symbolic reasoning with computational semantics. One research direction aims to directly apply the methods and techniques specific to symbolic approaches to computational models. For example, the work of Datta *et al.* [2] investigates Protocol Security Logic that is a variant of the PCL logic of Chapter *"Protocol Composition Logic"*. The crucial difference is that the semantics domain of the logic is a standard computational model, so logical statements are statements about such models. The work of Courant et al [25] is along similar lines: they show how to use a Hoare-like logic to reason directly about the computational security of various encryption schemes.

A related but different approach aims to formalize directly computational proofs. Examples include the work of Blanchet and Pointcheval [15] who develop a framework based on the game playing technique of Shoup [35] and Bellare and Rogaway [7], and that of Barthe *et al.* [5] who used Coq to develop a formal theory of the mathematics needed for cryptographic proofs, and apply it to several important cryptographic primitives.

Computational soundness. The second research direction is to determine if and under what conditions proofs carried out in symbolic models imply security under computational models. This direction became to be known as *computational soundness*. The first such result was due to Abadi and Rogaway [1] who prove a computational soundness result for security against passive adversaries. Techniques for dealing with the active case were later proposed by Backes, Pfitzmann, and Waidner [3] and by Micciancio and Warinschi [28]. Numerous other results have built upon these works by considering variations in the underlying execution model, the set of cryptographic primitives, and the range of security properties treated (see [20] for a comprehensive survey of the more recent work.)

At a high level, each of these results relies on the idea that the symbolic abstraction does not lose too much information regarding the actual (computational) execution. An alternative interpretation is that these results show that certain adversarial events that are

impossible to achieve in abstract models are highly improbable to achieve in computational ones.

In turn, this relation between symbolic and computational executions entails *computational soundness theorems*: a protocol that is secure symbolically, is also secure computationally. Since the same security property may be expressed in very different ways in the two settings, this second step does not immediately follow from the previous kind of results. However, when possible to establish, computational soundness theorems may prove truly valuable as they essentially say that computational security can be proved working entirely in symbolic models. In this chapter we give in detail a typical computational soundness result.

1.3. An extension of the Abadi-Rogaway logic with Diffie-Hellman keys

We describe an extension of the the work by Abadi and Rogaway [1]. There are several reasons that motivate our choice. Abadi and Rogaway's seminal result illustrates quite well the potential of the computational soundness approach. The simpler setting of a passive adversary enables relatively simple proofs, and the particular extension that we analyze raises some problems typical to computational soundness theorems. Furthermore, the result itself is applicable to realistic protocols.

In brief, the setting considered by Abadi and Rogaway [1] concerns protocols where parties exchange messages built from a set of basic messages (constants and keys) using concatenation and symmetric encryption. These messages are interpreted as either symbolic terms or as computational distributions on bit-strings. The soundness result of the paper is that symbolic terms that are equivalent (via an appropriate equivalence relation) have associated distributions that are computationally indistinguishable. This result implies that one can reason *symbolically* about indistinguishability of *computational* distributions.

The extension that we present considers an extended class of protocols where symmetric encryptions keys may also be derived via (general) Diffie-Hellman exchanges. In this setting parties share some cryptographic group and the set of protocol messages is extended with group elements. Furthermore, symmetric encryption keys can be derived from group elements using some key-derivation function. We show how to obtain the analogue of the Abadi-Rogaway result for this extended setting.

A generalization of the Diffie-Hellman assumption. The usefulness of computational soundness results depends on the size of the class of protocols that they treat. Trivial soundness theorems can be easily established. For example, one can simply limit the symbolic analysis to the empty class of protocols or, equally unsatisfactory, one can declare all protocols symbolically insecure. The difficulties lie in proving soundness for (non-trivial) symbolic analysis of large class of protocols.

One typical problem that often arises is that the intuition that guides the axiomatic security for primitives is not always warranted by the computational security definitions. More specifically, some uses of primitives may be considered secure when analyzed symbolically, but the same uses lead to complete breakdowns in computational models. It is therefore crucial that the design of the symbolic model requires to classify all possible uses of the primitives into secure and insecure.

Instantiated in the particular context of the extension that we present in this chapter the problem is as follows. We consider group elements obtained by exponentiating a

group generator to various polynomials. These elements can then be used as symmetric encryption keys. To create a symbolic model that is sound with respect to the computational model, it is crucial to understand which of the keys derived from group elements can be recovered and which keys are secret. Simple uses are easy to deal with. For example, a Diffie-Hellman key of the form $\mathsf{g}^{x_1 x_2}$ is secure even if g^{x_1} and g^{x_2} are known, however the key $\mathsf{g}^{x_1+x_2}$ can be easily computed. More generally however, for any arbitrary polynomial p, one has to determine precisely determine which keys of the form $\mathsf{g}^{p(x_1,x_2,\ldots,x_n)}$ can be recovered and which cannot.

In the first part of the chapter we present a generalization of the DDH assumption that accomplishes precisely this task. This result is entirely in the realm of computational cryptography and is absolutely necessary for developing the soundness result that we present in detail in the remainder of the chapter.

2. Notation and cryptographic preliminaries

In this section we introduce some notation and recall some of the main cryptographic notions necessary for our results.

We write $[k]$ for the set $\{1, 2, \ldots, k\}$. We write $|S|$ for the size of set S. If $\mathcal{A}$ is a randomized algorithm we write $x \xleftarrow{\$} \mathcal{A}(y)$ for the process of sampling from the distribution of $\mathcal{A}$'s output on input y. We write $\mathcal{A}^{\mathcal{O}}$ for an algorithm $\mathcal{A}$ that may query oracle $\mathcal{O}$ in its computation. Access to more than one oracle is indicated in the straightforward way.

2.1. Computational indistinguishability

Computational indistinguishability of distributions is a standard notion of equivalence between probability distributions on bitstrings. Informally, two probability distributions are indistinguishable if no efficient algorithm can determine from which of the two distributions a given bitstring has been sampled. The formal definition that we give below (as well as the definitions for other primitives and concepts that we utilize in this chapter) is in the style of concrete security. Recall that this style departs from that of asymptotic security (where security is guaranteed as a security parameter tends to infinity). Instead, concrete bounds on the running time of the adversary are translated into concrete bounds on the probability that the adversary is able to accomplish a specific task. One benefit is that one can fix a security parameter and thus avoid working with families of primitives. For example, in this setting one can compare probability distributions, as opposed to distribution *ensembles* (families of probability distributions indexed by the security parameter) as in the asymptotic setting.

Definition 1 (Computational indistinguishability) *Let X and Y be two probability distributions. We say that X and Y are (ϵ, t)-indistinguishable and we write $X \approx_{\epsilon,t} Y$, if, for any probabilistic distinguisher $\mathcal{D}$ that runs in time at most t, its advantage in distinguishing between the two distributions is smaller than ϵ:*

$$\mathbf{Adv}^{\mathsf{ind}}_{X,Y}(\mathcal{D}) = \left| \Pr[x \xleftarrow{\$} X \ : \ \mathcal{D}(x) = 1] - \Pr[y \leftarrow Y \ : \ \mathcal{D}(y) = 1] \right| < \epsilon$$

The probability is over the choice of x and y and the random coins of $\mathcal{D}$.

2.2. *Secure symmetric encryption schemes*

A symmetric encryption scheme is given by three algorithms:

$$\mathcal{KG} : \mathsf{Parameters} \times \mathsf{Coins} \to \mathsf{Keys}$$
$$\mathcal{E} : \mathsf{Keys} \times \mathsf{Plaintexts} \times \mathsf{Coins} \to \mathsf{Ciphertexts}$$
$$\mathcal{D} : \mathsf{Keys} \times \mathsf{Ciphertexts} \to \mathsf{Plaintexts} \cup \{\perp\}$$

The key generation algorithm $\mathcal{KG}$ is probabilistic. It takes as input a security parameter η and outputs a key k. We write $k \xleftarrow{\$} \mathcal{KG}(\eta)$ for the process of generating key k on input security parameter η. If the security parameter is fixed, or understood from the context, we may simply omit it and write $k \xleftarrow{\$} \mathcal{KG}$ for the same process. Notice that here as well as for all other randomized algorithms we do not explicitly show the dependence on the random coins used by the key generation algorithm. The encryption algorithm is also randomized. It takes as input a key k and a message $m \in \mathsf{Plaintexts}$ and returns a ciphertext c. We write $c \xleftarrow{\$} \mathcal{E}(k, m)$ for the process of generating ciphertext c, an encryption of plaintext m under key k. The decryption algorithm is typically deterministic. It takes as input a key k and a ciphertext c and returns the underlying plaintext or $\perp$ if the decryption process does not succeed. We write $m \leftarrow \mathcal{D}(k, c)$ for the process of obtaining m as a decryption of c using key k. It is mandated that for any k that can be the output of the key generation algorithm and for any message $m \in \mathsf{Plaintexts}$ it holds that $\mathcal{D}(k, \mathcal{E}(k, m)) = m$.

The security of encryption scheme comes in many flavors. We follow the definitional approach of Bellare et al. [8] and of Abadi and Rogaway [1]. The idea is to define security of an encryption scheme via the requirement that an adversary is not able to tell apart encryptions of messages that he chooses, from encryptions of some fixed string. One way to model this intuition is to consider adversaries that interact with an oracle to which it can submit arbitrary bitstrings. This oracle is keyed by some key k of the symmetric encryption scheme and can behave in one of two possible ways. The first possibility is that it returns encryptions under k of the bitstrings that the adversary submits as queries. We write $\mathcal{E}(k, \cdot)$ for this oracle to which we refer to as the "real" oracle. The second possible behavior is that the oracle ignores its query and simply returns an encryption of 0 under k. We write $\mathcal{E}(k, 0)$ for this oracle, and refer to it as the "fake" oracle. The task of the adversary is to figure out with which of the two possible oracles it interacts. Without loss of generality we may assume that the output of the adversary is binary: 1 for the guess that it interacts with the real oracle and 0 for the guess that it interacts with the fake oracle. A scheme is secure if the adversary cannot tell apart the two oracles, or in other words, that the adversary outputs 1 when it interacts with the first oracle about as often as when it interacts with the second oracle. Since in his attempt to distinguish between the two oracles the adversary gets to choose which plaintexts it wants to see encrypted, the resulting security notion is called indistinguishability against plaintext attack. Our presentation of the notion departs from the standard notion in that it requires that an adversary cannot tell apart encryptions of messages, even if these messages are of

different length. We write IND-CPA* for this notion to distinguish it from the standard IND-CPA notion[1].

Definition 2 (IND-CPA* security) *Let $\Pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme and η some fixed security parameter. We define the advantage of an adversary $\mathcal{A}$ by*

$$\mathbf{Adv}_{\Pi}^{\mathsf{ind\text{-}cpa}^*}(\mathcal{A}) = \\ = \left| \Pr\left[k \xleftarrow{\$} \mathcal{KG} \ : \ \mathcal{A}^{\mathcal{E}(k,\cdot)} = 1\right] - \Pr\left[k \xleftarrow{\$} \mathcal{KG} \ : \ \mathcal{A}^{\mathcal{E}(k,0)} = 1\right] \right|.$$

We say that an encryption scheme $(\mathcal{KG}, \mathcal{E}, \mathcal{D})$ is (ϵ, t)-IND-CPA*-secure if for any adversary $\mathcal{A}$ running in time t it holds that: $\mathbf{Adv}_{\Pi}^{\mathsf{ind\text{-}cpa}^*}(\mathcal{A}) < \epsilon$.

2.3. The Decisional Diffie-Hellman assumption

Fix a cyclic group $\mathbb{G}$ of order q and let g be a generator of $\mathbb{G}$. Informally, the Decisional Diffie-Hellman assumption states that it is difficult to distinguish a triple (g^{x_1}, g^{x_2}, g^r) from $(g^{x_1}, g^{x_2}, g^{x_1 x_2})$ when x_1, x_2, r are chosen uniformly at random.

Definition 3 (The Decisional Diffie-Hellman assumption) *Let $\mathbb{G}$ be a group of order q and $\mathfrak{g}$ a generator of $\mathbb{G}$. For an algorithm $\mathcal{A}$ define its advantage against the Decisional Diffie-Hellman problem in $\mathbb{G}$ by*

$$\mathbf{Adv}_{\mathbb{G}}^{\mathsf{ind\text{-}ddh}}(\mathcal{A}) = \left| \Pr\left[x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q \ : \ \mathcal{A}(\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{x_1 x_2}) = 1\right] \right. \\ \left. - \Pr\left[x_1, x_2, r \xleftarrow{\$} \mathbb{Z}_q \ : \ \mathcal{A}(\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{r}) = 1\right] \right|$$

The probabilities are taken over the choices of x_1, x_2 and r as well as the random coins of $\mathcal{A}$. We say that the (ϵ, t)-DDH assumption holds in $\mathbb{G}$, or alternatively that the DDH problem is (ϵ, t) hard in $\mathbb{G}$ if for all probabilistic adversaries $\mathcal{A}$ that run in time t it holds that $\mathbf{Adv}_{\mathbb{G}}^{\mathsf{ind\text{-}ddh}}(\mathcal{A}) < \epsilon$.

3. A generalization of the Decisional Diffie-Hellman assumption

In this section we present a generalization of the standard Decisional Diffie-Hellman assumption and explain its relation with the original problem.

3.1. Motivation

We start with a motivating discussion and some intuition. Consider some cyclic group $\mathbb{G}$ of order q, $\mathfrak{g}$ a generator of $\mathbb{G}$, and a tuple $t_1 = (\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{x_3}, \mathfrak{g}^{x_1 x_2 x_3})$, where the exponents are chosen uniformly at random. Can an adversary tell the tuple above from

[1] To obtain the standard IND-CPA security notion one replaces in the following definition the oracle $\mathcal{E}(k, 0)$ with the oracle $\mathcal{E}(k, 0^{|\cdot|})$ that when queried with message m returns the encryption of the all 0 string of length $|m|$.

the tuple $t_2 = (\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{x_3}, \mathfrak{g}^{r})$, i.e. the tuple where the last element is replaced with a group element chosen uniformly at random? It turns out that telling the two tuples apart is as hard as solving the DDH problem. This can be seen as follows. Consider the tuple $t_3 = (\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{x_3}, \mathfrak{g}^{rx_3})$, with randomly chosen x_1, x_2, x_3, r. Notice that t_3 is in fact selected from the same distribution as t_2 (the exponent in the last element of the tuple is chosen at random from $\mathbb{Z}_q$ in both cases and the rest of the components are the same). Finally, observe that an adversary $\mathcal{D}$ that distinguishes between t_1 and t_3 can be easily converted into an adversary that solves the DDH problem in $\mathbb{G}$. Indeed, on input a triple (a, b, c) adversary $\mathcal{A}$ selects $x_3 \xleftarrow{\$} \mathbb{Z}_q$ and passes to $\mathcal{D}$ as input the tuple (a, b, g^{x_3}, c^{x_3}). If the input of $\mathcal{A}$ is a true DDH tuple then the input of $\mathcal{D}$ follows the distribution of t_1. If the input of $\mathcal{A}$ is a fake DDH tuple, then the input to $\mathcal{D}$ follows the distribution of t_3.

Consider now the tuples $(\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{x_1+x_2})$ and $(\mathfrak{g}^{x_1}, \mathfrak{g}^{x_2}, \mathfrak{g}^{r})$. Clearly, in this case an adversary can immediately tell the two cases apart by simply checking if the product of the first two components of the tuple equals the third component. More generally, the adversary can tell if the exponent in the third component of the tuple is a linear combinations of the exponents of the first two components by raising these two appropriate powers and then multiplying them.

The two examples show that while for an adversary it is trivial to identify linear relations between the exponents observing any other sort of relation seems to be difficult.

In this section we confirm this intuition in the form of a powerful generalization of the DDH assumption. Informally, we allow the adversary to see $\mathfrak{g}^{p_i(x_1,x_2,\ldots,x_n)}$ for polynomials p_i in a set P, for randomly chosen $x_1, x_2, \ldots, x_n$. The task of the adversary is then to distinguish $\mathfrak{g}^{q_j(x_1,x_2,\ldots,x_n)}$ for polynomials q_j in some set Q from $\mathfrak{g}^{r_j}$ for randomly chosen r_j. It turns out that as long as the polynomials in Q are linearly independent and are not in the linear span of P then this task is tightly related to solving the DDH problem in the underlying group.

In the remainder of the section we formalize the generalization of the DDH problem and explain how it relates to the DDH assumption in the underlying group. The formal proof of the result is outside the scope of this chapter (see [13] for details), but we recall sufficiently many details to clarify the relation between the above problem and the DDH problem.

Fix a set of variables $\{X_1, X_2, \ldots, X_n\}$ and let $\mathbb{Z}_q[X_1, X_2, \ldots, X_n]$ be the ring of multivariate polynomials with coefficients in $\mathbb{Z}_q$. For a polynomial $p \in \mathbb{Z}_q[X_1, X_2, \ldots, X_n]$ we write $var(p)$ for the set of variables that occur in p and $mon(p)$ for the set of monomials (recall that a monomial is just a product of variables to positive integer powers) that occur in p. The notation extends to sets of polynomials in the obvious way. For example, for set $P = \{X_1X_3 + X_1X_4, X_2 + X_1X_4\}$ we have that $var(P) = \{X_1, X_2, X_3, X_4\}$ and $mon(P) = \{X_2, X_1X_3, X_1X_4\}$. XXX For a monomial m we write $ord(m)$ for the order of m (i.e., the sum of the powers of its variables). We say p is *power-free* if any $X_i \in var(p)$ occurs in p only raised to power 1. For example polynomials $X_1 + X_2X_3$ and $X_1X_2X_3$ are power-free but $X_1 + X_2^2$ is not. We write $\mathsf{PF}(\mathbb{Z}_q[X_1, X_2, \ldots, X_n])$ for the set of *power-free polynomials* with variables $\{X_1, \ldots, X_n\}$ and coefficients in $\mathbb{Z}_q$. The results that we present only hold for such polynomials, so in the remainder of the chapter, otherwise specified, the polynomials that we consider are power-free. For a set of polynomials $P = \{p_1, p_2, \ldots, p_n\}$ we write $\mathsf{Span}(P)$ for the set of polynomials spanned by P, that is

$$\mathsf{Span}(P) = \{\sum_{i=1}^{n} \lambda_i . p_i \mid \lambda_i \in \mathbb{Z}_q, i = 1, 2, \ldots, n\}$$

*3.2. The (P,Q)-*DDH *assumption*

In what follows we fix sets $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$ of power-free polynomials in $\mathbb{Z}_q[X_1, X_2, \ldots, X_n]$. Let $R = \{R_1, R_2, \ldots, R_m\}$ be a set of variables that do not occur in P and Q: $R \cap (var(P) \cup var(Q)) = \emptyset$. To these sets we associate three oracles that share an internal state which is a mapping $\tau : \{X_1, X_2, \ldots, X_n\} \cup R \to \mathbb{Z}_q$. We write $\mathsf{P}(\tau)$, $\mathsf{Q}(\tau)$ and $\mathsf{R}(\tau)$ for these oracles. The behavior of the oracles is as follows.

- Oracle P accepts as input any polynomial $p \in P$ and returns the value $\mathfrak{g}^{p(\tau(X_1),\tau(X_2),\ldots,\tau(X_n))}$.
- Oracles Q and R accept as input any polynomial $q_j \in Q$ and return, respectively, the values $\mathfrak{g}^{q_j(\tau(X_1),\tau(X_2),\ldots,\tau(X_n))}$ and $\mathfrak{g}^{\tau(R_j)}$.

An adversary is given access to either the pair of oracles P, Q, or to the pair of oracles P, R, and his goal is to determine which is the case. We call the problem associated to polynomial sets P and Q, the (P, Q)-DDH problem, and the pair (P, Q) a challenge. For example, if $P = \{X_1, X_2\}$ and $Q = \{X_1 X_2\}$ then the $(P, Q) -$ DDH assumption is precisely the DDH assumption. With other choices for P and Q one can easily get other variations of the DDH problem encountered throughout the literature. These include the group Diffie-Hellman assumptions [12], the parallel Diffie-Hellman assumption [11], the General Diffie-Hellman Exponent (GDHE) [10], the Square Exponent [26,19,36], the Inverse Exponent [33] and many others.

The assumption is given in the next definition.

Definition 4 (The (P, Q)-Decisional Diffie-Hellman assumption) *Let $\mathbb{G}$ be a cyclic group of order q and $\mathfrak{g}$ a generator of $\mathbb{G}$. Let $P, Q\mathsf{PF}(\subseteq \mathbb{Z}_q[X_1, X_2, \ldots, X_n])$ be finite sets of power-free polynomials. We define the advantage of an algorithm $\mathcal{A}$ against the (P, Q)-*DDH *problem by*

$$\mathbf{Adv}_{\mathbb{G}}^{(P,Q)\text{-}\mathsf{ddh}}(\mathcal{A}) = \left| \Pr\left[\mathcal{A}^{\mathsf{P}(\tau),\mathsf{Q}(\tau)} = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{P}(\tau),\mathsf{R}(\tau)} = 1\right] \right|$$

*where the probabilities are taken over the coins of $\mathcal{A}$ and over the map τ selected uniformly at random from the maps from $\{X_1, X_2, \ldots, X_n\} \cup R \to \mathbb{Z}_q$. We say that the (ϵ, t)-(P, Q)-*DDH *assumption holds in $\mathbb{G}$ or, alternatively, that the (P, Q)-*DDH *assumption is (ϵ, t)-hard in $\mathbb{G}$, if for any probabilistic adversary $\mathcal{A}$ that runs in time t, its advantage is at most ϵ.*

For convenience of notation, we sometimes write $\mathsf{Real}_{P,Q}$ and $\mathsf{Fake}_{P,Q}$ for the pairs of oracles (P, Q) and (P, R), respectively. If there is no ambiguity, we further simplify the notation to just Real and Fake.

The next lemma captures a class of pairs (P, Q) for which the the (P, Q)-DDH problem is easy to solve in any group.

Lemma 1 (Trivial challenges) *The (P, Q)-DDH problem is trivial to solve if at least one of the following conditions holds*

1. $\mathsf{Span}(P) \cap \mathsf{Span}(Q) \neq \{0\}$
2. *The polynomial in Q are linearly dependant*

If a challenge is not trivial we call it non-trivial. Next we aim to show that solving a non-trivial challenge is related to solving the Decisional Diffie-Hellman problem. We present a reduction that constructs an adversary against the latter problem from any adversary against the (P, Q)-DDH for any non-trivial challenge (P, Q). To specify how the success of the two adversaries are related we need to define several quantities. The following definitions from [13] are somewhat technical and can be safely skipped.

Define the graph $G_{(P,Q)}$ whose vertexes are the elements of $mon(P \cup Q)$, the monomials that occur in the polynomials of P and Q. There is an edge between two vertexes m_1 and m_2 if there exists $p \in P$ such that $m_1, m_2 \in mon(p)$. Let $mon^+_P(Q)$ be the set of monomials reachable in this graph from $mon(Q)$. The *order* of Q within P, denoted by $ord^+_P(Q)$, is defined by

$$ord^+_P(Q) = \sum_{m \in mon^+_P(Q)} \big(ord(m) - 1\big)$$

The set $\mathsf{nm}(P, Q)$ of non-maximal elements of $mon^+_P(Q)$ is the set of monomials $m \in mon^+_P(Q)$ such that there exists a strict super-monomial[2] m' of m that is in $mon(P)$ but not in $mon^+_P(Q)$. These two quantities are involved in the following relation between the DDH and (P, Q)-DDH problems. The following theorem says that an adversary that solves the (P, Q)-DDH problem can be used to solve the DDH problem (with some loss of efficiency that is quantified in the theorem).

Theorem 1 (Relating (P, Q)-DDH to DDH) *Let $\mathbb{G}$ be a cyclic group. Let (P, Q) be a non-trivial challenge over variables X_1 to X_n. If the* DDH *problem is (ϵ, t)-hard in $\mathbb{G}$, then (P, Q)-*DDH *is (ϵ', t')-hard in $\mathbb{G}$, for*

$$\epsilon' = 2\epsilon \cdot \Big(ord^+_P(Q) + \big(2^{|\mathsf{nm}(P,Q)|} - 1\big).\big(n + ord^+_P(Q)\big)\Big) \quad \textit{and} \quad 2t' = t.$$

The concrete bounds in the above theorem show that in some cases the loss of security can be exponential (when P and Q are such that the set $\mathsf{nm}(P, Q)$ is very large). Nevertheless, in most cases (e.g. when P contains only monomials) the loss is very small. In the remainder of the paper we will not be concerned with the precise bounds.

4. A Symbolic Logic for Diffie-Hellman Exponentials and Encryption

In this section we develop the soundness result of this paper. Recall that the goal is to show that reasoning about protocol messages using abstract, symbolic representations has meaningful implications with respect to computational models for such messages.

[2] This means that all the variables of m appear in m' and m is different from m'.

4.1. Syntax

We consider protocols where messages are constructed from a set of basic atoms: group elements, symmetric keys, and random nonces. More complex messages are obtained from basic messages using concatenation and encryption.

Formally, we fix finite but arbitrarily large sets of nonce symbols $\mathbf{Nonce} = \{N_1, N_2, \ldots\}$, keys for a symmetric encryption scheme $\mathbf{SKeys} = \{K_1, K_2, \ldots\}$, and exponents $\mathbf{Exponents} = \{X_1, X_2, \ldots\}$. We write $\mathbf{Poly}$ for the set of power-free polynomials over variables in $\mathbf{Exponents}$ with coefficients in $\mathbb{Z}_q$. Notice that q is an arbitrary but fixed natural number, and is a parameter of the symbolic language that we define here. Messages are terms in the set $\mathbf{Msg}$ defined by the grammar:

$$\begin{aligned}
\mathbf{Basic} &::= \mathbf{SKeys} \mid \mathsf{g}^{\mathbf{Poly}} \mid \mathbf{Nonce} \\
\mathbf{Keys} &::= \mathbf{SKeys} \mid\ h(\mathsf{g}^{\mathbf{Poly}}) \\
\mathbf{Msg} &::= \mathbf{Basic} \mid (\mathbf{Msg}, \mathbf{Msg}) \mid \{\!|\mathbf{Msg}|\!\}^{\mathsf{s}}_{\mathbf{Keys}}
\end{aligned}$$

In the above definition g and h are symbols different form all other symbols. They represent a generator for some group and a key derivation function, respectively. The second defining rule for $\mathbf{Keys}$ indicates that we consider encryption keys that are either generated by the key generation algorithm of an encryption scheme, or are derived from group elements using the key derivation function.

When an element K of $\mathbf{Keys}$ occurs in an expression of the form $\{\!|t|\!\}^{\mathsf{s}}_K$ is said to be in a *key position*. All other occurrences are said to be in *non-key* or *message position*.

In the remainder of the paper we assume that the pairing operation $(\_,\_)$ is associative so, for example, we simply write $(\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_3})$ instead of $(\mathsf{g}^{X_1}, (\mathsf{g}^{X_2}, \mathsf{g}^{X_3}))$. Furthermore, we may also omit the parenthesis. We refer to elements of $\mathbf{Msg}$ as terms, expressions, or messages interchangeably.

4.2. Symbolic analysis

In this section we capture symbolically the information that an adversary can obtain from messages. The conceptual ideas extend those of Abadi and Rogaway [1] and are as follows. First we fix the powers granted to the adversary. For example, the adversary should be able to decrypt ciphertexts for which it can obtain the decryption key. Next, to each term in $\mathbf{Msg}$ we attach a *pattern*. This is an (extended) expression that contains only the information an adversary can observe, given the powers that we grant him. The details follow.

THE SYMBOLIC ADVERSARY. The messages that an adversary can compute from a given set of terms are captured via a deduction relation $\vdash \subseteq \mathcal{P}(\mathbf{Msg}) \times \mathbf{Msg}$. For $S \subset \mathbf{Msg}$ and $E \in \mathbf{Msg}$, we write $S \vdash E$ to mean that the adversary can compute E given S. The deduction relation is the smallest relation defined by the rules in Figure 1.

The first four rules are straightforward. The rules on the second row indicate that the adversary knows the group generator, that it can multiply group elements that it knows, and that it can raise group elements to arbitrary (constant) powers. The first rule of the third row allows the adversary to decrypt a ciphertext under a key derived from a group

$$\frac{}{S \vdash E}\, E \in S \qquad \frac{S \vdash (E_1, E_2)}{S \vdash E_1} \qquad \frac{S \vdash (E_1, E_2)}{S \vdash E_2} \qquad \frac{S \vdash E_1, S \vdash E_2}{S \vdash (E_1, E_2)}$$

$$\frac{}{S \vdash \mathsf{g}^1} \qquad \frac{S \vdash \mathsf{g}^p, S \vdash \mathsf{g}^q}{S \vdash \mathsf{g}^{\lambda p + q}}$$

$$\frac{S \vdash \{\!|E|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}, S \vdash \mathsf{g}^p}{S \vdash E} \qquad \frac{S \vdash \{\!|E|\!\}^{\mathsf{s}}_{K}, S \vdash K}{S \vdash E}$$

Figure 1. Rules for defining the deduction power of the adversary. Symbols p and q range over $\mathbf{Poly}$, λ ranges over $\mathbb{Z}_q$ and K ranges over $\mathbf{SKeys}$.

$$\begin{array}{ll}
\mathsf{pattern}(E') = E' & \text{if } E' \in \mathbf{Basic} \\
\mathsf{pattern}\big((E_1, E_2)\big) = \big(\mathsf{pattern}(E_1), \mathsf{pattern}(E_2)\big) & \\
\mathsf{pattern}\big(\{\!|E'|\!\}^{\mathsf{s}}_{K}\big) = \{\!|\mathsf{pattern}(E')|\!\}^{\mathsf{s}}_{K} & \text{if } E \vdash K \\
\mathsf{pattern}\big(\{\!|E'|\!\}^{\mathsf{s}}_{K}\big) = \{\!|\Box|\!\}^{\mathsf{s}}_{K} & \text{if } E \nvdash K \\
\mathsf{pattern}\big(\{\!|E'|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}\big) = \{\!|\mathsf{pattern}(E')|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} & \text{if } E \vdash \mathsf{g}^p \\
\mathsf{pattern}\big(\{\!|E'|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}\big) = \{\!|\Box|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} & \text{if } E \nvdash \mathsf{g}^p
\end{array}$$

Figure 2. Rules for defining the pattern of expression E. In the above p ranges over $\mathbf{Poly}$ and K ranges over $\mathbf{SKeys}$.

element, provided that the adversary can compute that group element. The last rule allows the adversary to decrypt ciphertexts provided that the adversary can also derive the key used for encryption.

Example 1 *For example we have that:*

$$\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \{\!|K|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1+X_2})} \vdash K \textit{ but } \mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \{\!|K|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1X_2})} \nvdash K$$

The first deduction holds because from g^{X_1} and g^{X_2} the adversary can compute $\mathsf{g}^{X_1+X_2}$, and hence the encryption key. In the second example the adversary cannot obtain K since although it knows g^{X_1} and g^{X_2}, the deduction rules do not allow it to compute $\mathsf{g}^{X_1X_2}$ which is required for decryption.

PATTERNS FOR EXPRESSIONS. The information that an adversary obtains from a symbolic term can be captured by a pattern [1,27]. Intuitively, the pattern of expression $E \in \mathbf{Msg}$ is obtained by replacing all its unrecoverable sub-expressions (those sub-expressions that occur encrypted under keys that the adversary cannot derive from E) by the symbol $\Box$ (undecryptable). For an expression $E \in \mathbf{Msg}$ its pattern is defined by the inductive rules in Figure 2.

Example 2 *For example, the pattern of the expression*

$$\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \{\!|K_1|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1 X_2})}, \{\!|K_2|\!\}^{\mathsf{s}}_{K_1}, \{\!|K_3|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1+X_2})}, \{\!|K_4|\!\}^{\mathsf{s}}_{K_3}$$

is

$$\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \{\!|\square|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1 X_2})}, \{\!|\square|\!\}^{\mathsf{s}}_{K_1}, \{\!|K_3|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1+X_2})}, \{\!|K_4|\!\}^{\mathsf{s}}_{K_3}$$

EXPRESSION EQUIVALENCE. We aim to define an equivalence relation $\equiv$ on the set of expressions such that if two expressions convey the same amount of information to an adversary then the expressions should be deemed equivalent. A natural candidate relation (given the intuition that governs our definition of patterns) is therefore defined as follows:

$$E_1 \equiv E_2 \text{ if and only if } \mathsf{pattern}(E_1) = \mathsf{pattern}(E_2).$$

This notion of equivalence is however too strict. For example expressions K_1 and K_2 (which both represent symmetric keys) should be considered equivalent, yet their patterns are different. A similar situation also holds for nonces. The solution is to relax the notion of equality and consider equivalent those expressions that have equal patterns, modulo a (bijective) renaming of key and nonce symbols. The two expressions above would have equal patterns modulo the renaming that sends K_1 to K_2.

A more subtle issue concerns the use of exponentials. Consider expressions $E_1 = (\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_1 X_2})$ and $E_2 = (\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_3})$. Intuitively, if the DDH assumption is true then the two expressions should be equivalent. Unfortunately their patterns are clearly different. The reason is that our notion of patterns does not capture the relations that an adversary may observe between the various group elements that occur in an expression. The fix that we adopt is to consider equality of patterns modulo injective renamings of polynomials that define group elements, as long as these renamings do not change the set of observable relations between group elements. We therefore want to identify expressions E_1 and E_2 above, but at the same time, expressions E_1 and $E_3 = (\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_1+X_2})$ should remain distinguishable. Indeed, there exists a linear dependency in E_3 that does not exist in E_1. To determine which renamings are appropriate we rely on the results concerning the DDH assumption that we have developed in Section 3. Recall that the intuition that governs those results says that an adversary can only observe linear dependencies between the various exponents. Motivated by this intuition, we introduce the concept of *linear dependence preserving injective* renaming for a set $P \subseteq \mathbf{Poly}$ of polynomials. Such a renaming is an injective map from P to $\mathbf{Poly}$ which preserves the linear dependencies already existing in P, and that does not introduce new ones.

Definition 5 (Linear dependence preserving renamings) *Let $P \subseteq \mathbf{Poly}$ be an arbitrary set of polynomials and $\sigma : P \to \mathbf{Poly}$ be an injective map. Then σ is said to be linear dependence preserving (ldp) if:*

$$(\forall p_1, p_2, \ldots, p_n \in P)(\forall a_1, ..., a_n, b \in \mathbb{Z}_q)\ \sum_{i=1}^{n} a_i \cdot p_i = b \Leftrightarrow \sum_{i=1}^{n} a_i \cdot \sigma(p_i) = b$$

For an expression $E \in \mathbf{Msg}$ we write $\mathsf{poly}(E)$ for the set of polynomials that occurs in E. Given a mapping $\sigma : \mathsf{poly}(E) \to \mathbf{Poly}$ we write $E\sigma$ (or $\sigma(E)$) for the expression E in which for all $p \in \mathsf{poly}(E)$ we replace each occurrence of g^p with $\mathsf{g}^{\sigma(p)}$.

Example 3 *If* $E = (\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_1X_2})$ *and* $\sigma : \mathsf{poly}(E) \to \mathbf{Poly}$ *the map defined by* $\sigma(X_1) = X_1, \sigma(X_2) = X_2$ *and* $\sigma(X_1X_2) = X_3$ *is* ldp *and* $E\sigma = (\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_3})$. *In contrast,* σ' *with* $\sigma'(X_1) = X_1, \sigma'(X_2) = X_2$ *and* $\sigma'(X_1X_2) = X_1 + X_2$ *is not* ldp *since:* $X_1 + X_2 + (q-1)X_1X_2 \neq \sigma'(X_1) + \sigma'(X_2) + (q-1)\sigma'(X_1X_2) = 0$

The following notion of expression equivalence accounts for bijective renamings of keys and nonces, as well as for the relations that an adversary may observe between the various group elements that occur in an expression via ldp renamings.

Definition 6 *Expressions* E_1 *and* E_2 *are equivalent, and we write* $E_1 \cong E_2$*, if there exists injective renamings* $\sigma_1 : \mathbf{SKeys} \to \mathbf{SKeys}$*,* $\sigma_2 : \mathbf{Nonce} \to \mathbf{Nonce}$*, and injective* ldp*-renaming* $\sigma_3 : \mathsf{poly}(E_2) \to \mathbf{Poly}$ *such that:*

$$\mathsf{pattern}(E_1) = \mathsf{pattern}(E_2\sigma_1\sigma_2\sigma_3)$$

ACYCLIC EXPRESSIONS. An important concept for computational soundness in general and for the results in this chapter in particular is the notion of acyclic expressions [1]. Informally, these are expressions where encryption cycles (situations where a key K_1 encrypts key K_2, which in turn encrypts K_3, and so on, but one of the keys K_i is equal to K_1) do not occur. Due to technical reasons, soundness cannot be guaranteed in such situations by standard assumptions on the encryption scheme.

For an expression E we define the relation $\prec_E$ on the set $\mathbf{SKeys} \cup \mathbf{Poly}$ (restricted to the symmetric keys and polynomials that actually occur in E). For any $u, v \in \mathbf{SKeys} \cup \mathbf{Poly}$ we set $u \prec_E v$ if u or g^u occurs in E in a message position encrypted with v (if $v \in \mathbf{SKeys}$) or with $h(v)$ (if $v \in \mathbf{Poly}$). This relation is used in defining formally acyclic expressions.

Definition 7 (Acyclic expression) *An expression* E *is* acyclic *if the two following conditions are satisfied:*

1. *If* p *is a polynomial such that* $h(\mathsf{g}^p)$ *occurs as an encryption key in* E*, then* p *is not a linear combination of the polynomials that occur in* E *(and are different from* p*).*
2. *The relation* $\prec_E$ *induced by* E *is acyclic.*

The first condition is intended to avoid encryptions in which the plaintext and the encryption key are linearly dependent, as for example in $\{|\mathsf{g}^{X_1}, \mathsf{g}^{X_1+X_2}|\}^{\mathsf{s}}_{h(\mathsf{g}^{X_2})}$. It can be easily shown that the occurrence of such a ciphertext can reveal the encryption key without contradicting the security of the encryption scheme. This condition can be further relaxed by requiring that p is not a linear combination of polynomials that occur encrypted under g^p. We work with the more restrictive but simpler condition of Definition 7 only for simplicity.

Algorithm pat(E)
$i \leftarrow 0$; $\mathcal{K} \leftarrow \mathsf{uKeys}(E)$; $n \leftarrow \mid \mathcal{K} \mid$; $E_i \leftarrow E$
while $\mathcal{K} \neq \emptyset$ do
select an element K maximal in $\mathcal{K}$ with respect to $\prec_E$
$E_{i+1} \leftarrow E_i[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]$
$\mathcal{K} \leftarrow \mathcal{K} \setminus \{K\}$
$i \leftarrow i + 1$
output E_n

Figure 3. Iterative algorithm for computing the pattern of an acyclic expression E.

For acyclic expressions we give in Figure 3 an algorithmic procedure for computing their associated pattern. The algorithm uses the set $\mathsf{uKeys}(E) = \{K \mid K \in \mathbf{Keys}, E \not\vdash K\}$, the set of keys that cannot be recovered from E. Notice that K ranges over both standard keys and keys derived from group elements. Informally, the algorithm works as follows. It selects an arbitrary unrecoverable key that does not occur encrypted in E and replaces all encryptions under that key with encryptions of $\Box$. We write $E[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]$ for the expression obtained from E by replacing each occurrence of an encryption $\{\!|t|\!\}^{\mathsf{s}}_K$ with $\{\!|\Box|\!\}^{\mathsf{s}}_K$. The algorithm repeats until there is no unrecoverable key in E that occurs in a message position. This final expression is the pattern of the original expression. The algorithm is formalized in Figure 3.

4.3. Computational Interpretation

Next we show how expressions can be interpreted, computationally, as distributions on bitstrings.

TERMS AS DISTRIBUTIONS. Computationally, each element of $\mathbf{Msg}$ (and more generally, any pattern) is interpreted as a distribution. For example, elements of $\mathbf{SKeys}$ represent (the distributions of) cryptographic keys obtained by running the key generation algorithm of some (fixed) encryption scheme. A term like g^X represents the distribution of $\mathfrak{g}^x$ when exponent x is chosen at random, and $h(\mathsf{g}^{X_1 X_2})$ represents the distribution of keys obtained by applying a key derivation function to the group element $\mathfrak{g}^{x_1 x_2}$ for random x_1 and x_2. A slightly more complex example is the expression: $(\mathsf{g}^x, \mathsf{g}^y, \{\!|K|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{xy})})$ that represents the distribution of a conversation between two parties that first exchange a Diffie-Hellman key, and then use this key to encrypt a symmetric key.

We now specify precisely how to associate to each symbolic expression in $\mathbf{Msg}$ a distribution. For this, we fix an encryption scheme $\Pi = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$, a group $\mathbb{G} = \langle \mathfrak{g} \rangle$ of order q, and a key-derivation function $\mathsf{kd} : \mathbb{G} \to \{0,1\}^\eta$. Here, and throughout the remainder of the section η is a fixed security parameter. About the key derivation function we assume that when applied to a random group element it returns a symmetric key, distributed identically with the output of the key generation function $\mathcal{KG}$.

The algorithm in Figure 4 associates to each expression in $\mathbf{Msg}$ (and in fact, more generally, to each pattern) a distribution. The association is as follows. First, the randomized algorithm Init is executed to obtain a map τ that associates to each key, nonce, and random exponent that occur in E a bitstring. The symbol $\Box$ is mapped to some fixed

Algorithm $\mathsf{Init}(E)$

For each key $K \in \mathbf{SKeys}$ set $\tau(K) \xleftarrow{\$} \mathcal{KG}(\eta)$

For each nonce $N \in \mathbf{Nonce}$ set $\tau(N) \xleftarrow{\$} \{0,1\}^\eta$

For each $X \in \mathbf{Exponents}$ set $\tau(X) \xleftarrow{\$} \{1, 2, \ldots, |\mathbb{G}|\}$

Set $\tau(\square) = 0$

Output τ

Algorithm $\mathsf{Gen}(E, \tau)$

If $E \in \mathbf{SKeys} \cup \mathbf{Nonce} \cup \{\square\}$ then $e \leftarrow \tau(E)$

If $E = (E_1, E_2)$ then $e_1 \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $e_2 \xleftarrow{\$} \mathsf{Gen}(E_2, \tau)$; $e \leftarrow \langle e_1, e_2 \rangle$

If $E = \mathsf{g}^{p(X_1, X_2, \ldots, X_n)}$ then $e \leftarrow \mathsf{g}^{p(\tau(X_1), \tau(X_2), \ldots, \tau(X_n))}$

If $E = \{\!|E_1|\!\}^{\mathsf{s}}_K$ then $m \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $e \xleftarrow{\$} \mathcal{E}(\tau(K), m)$

If $E = \{\!|E_1|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}$ then $m \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $k \leftarrow \mathsf{Gen}(\mathsf{g}^p, \tau)$ $e \xleftarrow{\$} \mathcal{E}(\mathsf{kd}(k), m)$

Return e

Figure 4. Algorithms for associating a distribution to an expression (or a pattern) E.

string, here 0. Then, algorithm $\mathsf{Gen}(E, \tau)$ uses τ to (recursively) compute a bit-string associated to each subterm of E, inductively. For example, expression (E_1, E_2) is mapped to $\langle e_1, e_2 \rangle$, where $\langle \cdot, \cdot \rangle : \{0,1\}^* \to \{0,1\}^*$ is some injective, invertible mapping which we fix, but leave unspecified, and e_1 and e_2 are sampled according to E_1 and E_2, respectively. Similarly, expression $\{\!|E_1|\!\}^{\mathsf{s}}_K$ is mapped to encryption of e_1 under $\tau(K)$, where e_1 is sampled according to E_1 and $\tau(K)$ is the key associated to K by τ. The interpretation of keys derived from group elements are obtained from the computational interpretation of those elements by using the key derivation function kd.

We write $[\![E]\!]_{\Pi,\mathbb{G}}$ for the distribution of e obtained using the two algorithms, that is: $\tau \xleftarrow{\$} \mathsf{Init}(E)$; $e \xleftarrow{\$} \mathsf{Gen}(E, \tau)$. Since Π and $\mathbb{G}$ are clear from the context, we may simply write $[\![E]\!]$ for the same distribution.

4.4. Soundness of symbolic analysis

In this section we show that symbolic analysis can be meaningfully used to reason about the distributions associated to expressions. Specifically, we establish two results. The first result states that the distribution associated to an expression and the distribution associated to its pattern are indistinguishable. Informally, this means that patterns are a sound abstraction of the information obtained by an adversary: no information is lost even if parts of the expression are essentially erased. Building on this result, we show that equivalent expressions (expressions that have the same pattern modulo appropriate renamings) also have indistinguishable distributions. Computational indistinguishability of distributions can therefore be established by reasoning entirely symbolically.

Before stating our soundness theorem we comment on the requirements that we put on the primitives used in the implementation. As described in Section 2 and similarly to the original paper of Abadi and Rogaway [1], we implement encryption using a scheme that is $\mathsf{IND\text{-}CPA}^*$. We emphasize that we use the additional requirement only for sim-

plicity – a similar result can be obtained for IND-CPA security, by refining the pattern definition [29,27].

The implementation that we consider uses a group where the DDH problem is hard. In particular, this implies that for any acyclic expression E if $h(\mathsf{g}^p)$ is an unrecoverable encryption key that occurs in E, then the (ϵ_p, t_p)-$(\mathsf{poly}(E) \setminus \{p\}, \{p\})$-DDH assumption also holds in $\mathbb{G}$, for some parameters ϵ_p, t_p that depend on the structure of $\mathsf{poly}(E)$ and on p. Indeed, if E is acyclic then p is not a linear combination of the other polynomials that occur in E. The results of this section require a group where each $(\mathsf{poly}(E) \setminus \{p\}, \{p\})$-DDH problem is difficult. To avoid enumerating over all polynomials in E, and to obtain a result independent of the structure of the polynomials we simply demand that the group used in the implementation ensures the required level of difficulty.

Definition 8 *Group $\mathbb{G}$ is said to be (ϵ, t)-appropriate for acyclic expression E if the $(\mathsf{poly}(E) \setminus \{p\}, \{p\})$-DDH problem is (ϵ, t)-hard in $\mathbb{G}$, for all p for which $h(\mathsf{g}^p)$ is an unrecoverable encryption key in E.*

Finally, we also require that the key derivation function kd is such that $\mathcal{KG}(\eta)$ and $\mathsf{kd}(\mathsf{g}^r)$ output equal distributions when r is selected at random.

We can now state and prove a first result that establishes a link between the symbolic and the computational models for expressions. Informally, it says that from the point of view of a computational adversary, no significant amount of information is lost if one replaces an acyclic expression with its pattern.

Theorem 2 *Let E be an acyclic expression. Let Π be an (ϵ_Π, t_Π)-IND-CPA* secure encryption scheme and $\mathbb{G}$ be an $(\epsilon_\mathbb{G}, t_\mathbb{G})$-appropriate group for E. Then,*

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_E, t_E} [\![\mathsf{pattern}(E)]\!]_{\Pi,\mathbb{G}}$$

where ϵ_E and t_E are parameters dependent on E that we determine in the proof.

Proof:
The proof makes use of the iterative way of computing patterns for acyclic expressions (Figure 3). Let $E_0, E_1, \ldots, E_n$ be the sequence of expressions computed during the execution of the algorithm. The expressions are such that $E_0 = E$ and $E_n = \mathsf{pattern}(E)$. First, notice that it is sufficient to show that distributions $D_i = [\![E_i]\!]$ and $D_{i+1} = [\![E_{i+1}]\!]$ are close for any i to conclude that $[\![E_0]\!]$ and $[\![E_n]\!]$ are also close. The following lemma formalizes this intuition.

Lemma 2 *Let $D_0, D_1, \ldots, D_n$ be distributions such that for any $i \in \{0, 1, \ldots, n-1\}$ it holds that $D_i \approx_{\epsilon_i, t_i} D_{i+1}$. Then $D_0 \approx_{\epsilon,t} D_n$ holds, where $\epsilon = n \cdot \max_i \epsilon_i$ and $t = \min_i t_i$.*

To see this, consider an adversary $\mathcal{A}$ that runs in time t. Then, we have that:

$$\begin{aligned}
&\left|\Pr\left[x \xleftarrow{\$} D_n : \mathcal{A}(x) = 1\right] - \Pr\left[x \xleftarrow{\$} D_0 : \mathcal{A}(x) = 1\right]\right| \leq \\
&\leq \sum_{i=1}^{n} \left|\Pr\left[x \xleftarrow{\$} D_i : \mathcal{A}(x) = 1\right] - \Pr\left[x \xleftarrow{\$} D_{i-1} : \mathcal{A}(x) = 1\right]\right| \\
&\leq n \cdot \max_i \epsilon_i
\end{aligned}$$

where the first inequality follows by triangle inequality and the last one from the assumption on the distributions.

Let $E_0, E_1, \ldots, E_n$ be the sequence of expressions computed by the algorithm. Recall that $E_{i+1} = E_i[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]$ for some $K \in \mathbf{Keys}$. We show that this transformation does not significantly change the distribution associated to E_i, provided that the primitives used in the implementation are secure. The desired relation between the distributions associated to $E = E_0$ and $\mathsf{pattern}(E) = E_n$ then follow from the argument outlined above.

To argue that distributions of $[\![E]\!]$ and $[\![E[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]]\!]$ are close we differentiate between the case when K is in $\mathbf{SKeys}$ and when K is in $h(\mathsf{g}^{\mathbf{Poly}})$. We give two lemmas. The first lemma directly treats the first case. The second lemma is a stepping stone for proving the second case. We first state these lemmas and explain how to use them to obtain our result. Their proofs are in Appendix A and Appendix B, respectively.

Lemma 3 *Let E be an acyclic expression, and $K \in \mathbf{SKeys}$ a key that only occurs in E in key positions. Then, if Π is an (ϵ_Π, t_Π)-IND-CPA* secure encryption scheme, then:*

$$[\![E]\!] \approx_{\epsilon,t} [\![E[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]]\!]$$

for $\epsilon = \epsilon_\Pi$ and $t = t_\Pi - t_{sim\mathcal{B}}$ for some constant $t_{sim\mathcal{B}}$.

The next lemma states that if in an expression E one replaces an encryption key $h(\mathsf{g}^p)$ with a random encryption key K_0 the distribution of the expression does not change significantly. In the statement of the lemma, and throughout the paper, we write $E[\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_K]$ for the expression that results by replacing each occurrence of encryption key $h(\mathsf{g}^p)$ by K.

Lemma 4 *Let E be an acyclic expression, $h(\mathsf{g}^{p_0})$ an encryption key that only occurs in E in key positions. Let $K_0 \in \mathbf{SKeys}$ be a key that does not occur in E. If $(\mathsf{poly}(E) \setminus \{p_0\}, \{p_0\})$-DDH is $(\epsilon_\mathbb{G}, t_\mathbb{G})$-difficult in $\mathbb{G}$ then:*

$$[\![E]\!] \approx_{\epsilon,t} [\![E[\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^{p_0})} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{K_0}]]\!]$$

for $\epsilon = \epsilon_\mathbb{G}$ and $t = t_\mathbb{G} - t_{sim\mathcal{C}}$, for some constant $t_{sim\mathcal{C}}$.

We now explain how to use the previous two lemmas to conclude that the distance between distributions E_i and E_{i+1} is small. If $E_{i+1} = E_i[\{\!|\ |\!\}^{\mathsf{s}}_K \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_K]$ and $K \in \mathbf{SKeys}$ it follows by Lemma 3 that:

$$[\![E_i]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_\Pi, (t_\Pi - t_{sim\mathcal{B}})} [\![E_{i+1}]\!]_{\Pi,\mathbb{G}} \tag{1}$$

The case when $E_{i+1} = E_i[\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}]$ is slightly more complex, as it is not directly covered by the above lemmas. Nevertheless, we can use the transformations in the lemmas to "emulate" the transformation above. Specifically, to simulate the result of the transformation $\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}$ on an expression one can apply, sequentially, the following three transformations: $\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{K_0}$ where $K_0 \in \mathbf{SKeys}$ is a key that does not occur in the expression, then the transformation $\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}$, followed by the transformation $\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}$. Essentially,

the first transformation replaces all occurrences of encryption key $h(\mathsf{g}^p)$ with a fresh encryption key K_0. The next transformation replaces all terms encrypted with K_0 by $\Box$. Finally, the key K_0 is transformed back into $h(\mathsf{g}^p)$. This can be seen as follows. Let $E_i^0 = E_i$, $E_i^1 = E_i[\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^{p_0})} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{K_0}]$, $E_i^2 = E_i^1[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]$, and $E_i^3 = E_i^2[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^{p_0})}]$. Notice that $E_i^3 = E_{i+1}$. Since each of the three transformations above is covered by one of the two lemmas above we obtain a relation between the distributions $[\![E_i]\!]$ and $[\![E_{i+1}]\!]$.

Since K_0 only occurs in E_i^1 as an encryption key, by Lemma 3, we have that

$$[\![E_i^1]\!] \approx_{\epsilon_\Pi, (t_\Pi - t_{sim\mathcal{B}})} [\![E_i^2]\!]$$

By Lemma 4, we have that:

$$[\![E_i^0]\!] \approx_{\epsilon_\mathbb{G}, (t_\mathbb{G} - t_{sim\mathcal{C}})} [\![E_i^1]\!]$$

and also

$$[\![E_i^3]\!] \approx_{\epsilon_\mathbb{G}, (t_\mathbb{G} - t_{sim\mathcal{C}})} [\![E_i^2]\!]$$

The last result follows since $E_2^i = E_3^i[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^{p_0})}]$. >From the above equations, by Lemma 2 we get that

$$[\![E_i]\!] = [\![E_i^0]\!] \approx_{\epsilon,t} [\![E_i^3]\!] = [\![E_{i+1}]\!]$$

where $\epsilon = \epsilon_\Pi + 2 \cdot \epsilon_\mathbb{G}$ and $t = \min((t_\Pi - t_{sim\mathcal{B}}), (t_\mathbb{G} - t_{sim\mathcal{C}}))$.

We have argued that for any $i \in \{0, 1, \ldots, n-1\}$, $[\![E_i]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_i, t_i} [\![E_{i+1}]\!]_{\Pi,\mathbb{G}}$ with $\epsilon_i = \epsilon_\Pi$ or $\epsilon_i = \epsilon_\Pi + 2 \cdot \epsilon_\mathbb{G}$, and $t_i \leq \min(t_\Pi - t_{sim\mathcal{B}}, t_\mathbb{G} - t_{sim\mathcal{C}})$, then by another application of Lemma 2 we obtain:

$$[\![E]\!]_{\Pi,\mathbb{G}} = [\![E_0]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_E, t_E} [\![E_n]\!]_{\Pi,\mathbb{G}} = [\![\mathsf{pattern}(E)]\!]_{\Pi,\mathbb{G}}$$

for $\epsilon_E \leq n \cdot (\epsilon_\Pi + 2 \cdot \epsilon_\mathbb{G})$ and $t_E = \min(t_\Pi - t_{sim\mathcal{B}}, t_\mathbb{G} - t_{sim\mathcal{C}})$, as desired.

The next theorem establishes the main result of this chapter: symbolically equivalent expressions have associated computationally indistinguishable distributions. The result holds for acyclic expressions as long as encryption is IND-CPA* and a more stringent requirement holds on the group $\mathbb{G}$. Namely, we require that if $p_1, p_2, \ldots, p_t$ is a base of $\mathsf{poly}(E)$ then the $(\{\}, \{p_1, p_2, \ldots, p_t\})$-DDH problem is hard in $\mathbb{G}$. We overload the meaning of "appropriateness", and in what follows we will assume that in addition to the requirements of Definition 8, the above problem is also (ϵ, t)-hard in $\mathbb{G}$.

Theorem 3 (Symbolic equivalence implies computational indistinguishability) *Let $\mathbb{G}$ be a group that is $(\epsilon_\mathbb{G}, t_\mathbb{G})$-appropriate for acyclic expressions E_1 and E_2 and Π an (ϵ_Π, t_Π)-IND-CPA* secure encryption scheme. Then*

$$E_1 \cong E_2 \implies [\![E_1]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon,t} [\![E_2]\!]_{\Pi,\mathbb{G}}$$

for some ϵ and t (that we determine in the proof).

The proof builds on Theorem 2. Since, by definition, the two expressions are equivalent if their patterns are related via bijective renamings of keys and nonces, and ldp-renamings of polynomials, it is sufficient to show that such renamings do not significantly change the distribution associated to a pattern. Before we give the proof we introduce two useful lemmas. The first lemma simplifies dealing with arbitrary polynomials in the exponents.

The lemma says that if $p_1, p_2, \ldots p_t$ is a basis of $\mathsf{poly}(E)$, then the ldp renaming which for each $i \in [t]$ maps p_i to some fresh variable Y_i (that does not occur in E) does not change the distribution associated to E.

Lemma 5 *Let E be an acyclic expression, $\{p_1, p_2, \ldots, p_t\}$ a base of $\mathsf{poly}(E)$, let $Y_1, Y_2, \ldots, Y_t$ be exponent variables that do not occur in E. Let σ be the ldp-renaming with $\sigma(p_i) = Y_i$. Let Π be an arbitrary symmetric encryption scheme and $\mathbb{G}$ a group that is $(\epsilon_{\mathbb{G}}, t_{\mathbb{G}})$-appropriate for E. Then*

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_{\mathbb{G}}, t_{\mathbb{G}} - t_{sim\mathcal{E}}} [\![E\sigma]\!]_{\Pi,\mathbb{G}}$$

Proof:
We show that an algorithm $\mathcal{D}$ that distinguishes between the distributions of E and that of $E\sigma$ breaks the assumptions that $\mathbb{G}$ is appropriate for E. More specifically, we show that a successful distinguisher $\mathcal{D}$ can be used to construct an adversary $\mathcal{E}$ against the the $(\{\}, \{p_1, p_2, \ldots, p_t\})$-DDH problem. The construction is given in Figure 5.

Adversary $\mathcal{E}^{\mathcal{O}_1, \mathcal{O}_2}(E)$
  $\tau \xleftarrow{\$} \mathsf{Init}(E)$
  For $i \in \{1, 2, \ldots, t\}$ do
    set $\tau(\mathsf{g}^{p_i}) \xleftarrow{\$} O_2(i)$
  $e \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}_1, \mathcal{O}_2}(E, \tau)$
  Return $\mathcal{D}(e)$

Algorithm $\mathsf{mGen}^{\mathcal{O}_1, \mathcal{O}_2}(E, \tau)$
  If $E \in \mathbf{SKeys} \cup \mathbf{Nonce} \cup \{\Box\}$ then $e \leftarrow \tau(E)$
  If $E = (E_1, E_2)$ then $e_1 \xleftarrow{\$} \mathsf{mGen}(E_1, \tau)$; $e_2 \xleftarrow{\$} \mathsf{mGen}(E_2, \tau)$; $e \leftarrow \langle e_1, e_2 \rangle$
  If $E = \mathsf{g}^{p(X_1, X_2, \ldots, X_n)}$ then
    let $\lambda_1, \lambda_2, \ldots, \lambda_t$ such that $p = \sum_{i=1}^{t} \lambda_i \cdot p_i$
    $e \leftarrow \sum_{i=1}^{n} \lambda_i \cdot \tau(\mathsf{g}^{p_i})$;
  If $E = \{E_1\}_K$ then $m \xleftarrow{\$} \mathsf{mGen}(E_1, \tau)$; $e \xleftarrow{\$} \mathcal{E}(\tau(K), m)$;
  If $E = \{E_1\}_{h(\mathsf{g}^p)}$ then
    let $\lambda_1, \lambda_2, \ldots, \lambda_t$ such that $p = \sum_{i=1}^{t} \lambda_i \cdot p_i$
    $k \leftarrow \sum_{i=1}^{n} \lambda_i \cdot \tau(\mathsf{g}^{p_i})$; $m \xleftarrow{\$} \mathsf{mGen}(E_1, \tau)$; $e \leftarrow \mathcal{E}(\mathsf{kd}(k), m)$
  Return e

Figure 5. Adversary $\mathcal{E}$ is against the $(\{\}, \{p_1, p_2, \ldots, p_t\})$-DDH assumption. It uses procedure mGen which makes use of the oracle of $\mathcal{E}$.

□

The adversary is against the $(\{\}, \{p_1, p_2, \ldots, p_t\})$-DDH problem and therefore has access to two oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ – the first oracle is trivial for this particular problem. The adversary $\mathcal{E}$ constructs a bitstring e much in the same way bitstrings are associated to expressions by algorithm Gen: it generates bitstring interpretation for all of the symbols that occur in E using the Init procedure. However from the resulting assignment it only uses the interpretation for keys and nonces. The interpretation for group elements uses the oracles: for each group element g^p, if $p = p_i$ for some $i = 1, 2, \ldots, t$ (that is, if p belongs to the base of $\mathsf{poly}(E)$), then its interpretation is obtained by querying p to $\mathcal{O}_2$. The interpretation of g^p for any other polynomial p is obtained by using the coefficients of the representation of p in base $\{p_1, p_2, \ldots, p_t\}$ in the straightforward way.

Consider now the two possibilities for the oracle $\mathcal{O}_2$. When this is the "real" oracle, the interpretations of group elements are precisely as in the Gen algorithm so e is distributed according to $[\![E]\!]$. When $\mathcal{O}_2$ is the "fake" oracle the interpretation of g^{p_i} is a random group element obtained by g^{r_i} where r_i is a randomly chosen exponent. In this case the sample e computed by $\mathcal{E}$ is from $[\![E\sigma]\!]_{\Pi,\mathbb{G}}$.

We therefore have that (to simplify notation in the following equation we write P for the empty set and Q for $\{p_1, p_2, \ldots, p_t\}$):

$$\begin{aligned}\mathbf{Adv}_{\mathbb{G}}^{(P,Q)\text{-DDH}} &= \Pr\left[\mathcal{E}^{\mathsf{Real}_{(P,Q)}=1}\right] - \Pr\left[\mathcal{E}^{\mathsf{Fake}_{(P,Q)}} = 1\right] \\ &= \Pr\left[e \xleftarrow{\$} [\![E]\!] : \mathcal{D}(e) = 1\right] - \Pr\left[e \xleftarrow{\$} [\![E\sigma]\!] : \mathcal{D}(e) = 1\right]\end{aligned}$$

As before, since the running time of $\mathcal{E}$, $t_{\mathcal{E}}$ is $t_{sim\mathcal{E}} + t_{\mathcal{D}}$, where $t_{sim\mathcal{E}}$ is the time that $\mathcal{E}$ takes to construct the sample it passes to $\mathcal{D}$, if the $(\{\}, \{p_1, p_2, \ldots, p_t\})$-DDH is $(\epsilon_{\mathbb{G}}, t_{\mathbb{G}})$-difficult, then it must be the case that $[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_{\mathbb{G}}, t_{\mathbb{G}} - t_{sim\mathcal{E}}} [\![E\sigma]\!]$, as desired. □

The next lemma is the core of the proof of Theorem 3. It states that injective renamings of nonces and keys, as well as injective ldp renamings do not significantly change the distribution of an expression.

Lemma 6 *Let E be an acyclic expression. Let σ_1 be a permutation on* **Keys***, σ_2 be a permutation on* **Nonces** *and σ_3 be a* ldp*-renaming of polynomials in* $\mathsf{poly}(E)$*. If $\mathbb{G}$ is appropriate for E, then*

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon,t} [\![E\sigma_1\sigma_2\sigma_3]\!]_{\Pi,\mathbb{G}}$$

where $\epsilon = 2 \cdot \epsilon_{\mathbb{G}}$ and $t = t_{\mathbb{G}} - t_{sim\mathcal{E}}$, for some constant $t_{sim\mathcal{E}}$.

Proof:
First, notice that it is trivial that any injective renaming of keys and nonces leaves the distribution of an expression unchanged. The reason is that the algorithm that associates distributions to bitstrings does not depend on the particular symbols used in the expression. It is therefore sufficient to show that for any acyclic expression E if $\sigma_3 : \mathsf{poly}(E) \to \mathbf{Poly}$ is an arbitrary ldp-renaming then E and $E\sigma_3$ have distributions that are indistinguishable:

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon,t} [\![E\sigma_3]\!]_{\Pi,\mathbb{G}}$$

Let $p_1, p_2, \ldots, p_t$ be a base of $\mathsf{poly}(E)$. Since σ_3 is ldp, then $\sigma_3(p_1), \sigma_3(p_2), \ldots, \sigma_3(p_t)$ is a base for $\mathsf{poly}(E\sigma_3)$. Consider the ldp-renaming $\sigma : \mathsf{poly}(E) \to \mathbf{Poly}$ that for each $i \in [t]$ maps p_i to Y_i. Then, $\sigma_3^{-1}; \sigma : \mathsf{poly}(E\sigma_3) \to \mathbf{Poly}$ is also an ldp-renaming that maps $\sigma_3(p_i)$ to Y_i. (Here we write $\sigma_1; \sigma_2$ for the renaming obtained by first applying σ_1 followed by σ_2.) The expression E and the substitution σ satisfy the conditions of Lemma 5, so it holds that:

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_{\mathbb{G}}, t_{\mathbb{G}} - t_{sim\mathcal{E}}} [\![E\sigma]\!]_{\Pi,\mathbb{G}} \tag{2}$$

Similarly, expression $E\sigma_3$ and ldp-renaming $\sigma_3^{-1}; \sigma$ satisfy the conditions of the same lemma so we get that

$$[\![E\sigma_3]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon_{\mathbb{G}}, t_{\mathbb{G}} - t_{sim\mathcal{E}}} [\![E\sigma_3^{-1}; \sigma_3; \sigma]\!]_{\Pi,\mathbb{G}} = [\![E\sigma]\!]_{\Pi,\mathbb{G}} \tag{3}$$

By the two equations above and Lemma 2 we conclude that

$$[\![E]\!]_{\Pi,\mathbb{G}} \approx_{2\cdot\epsilon_{\mathbb{G}}, t_{\mathbb{G}} - t_{sim\mathcal{E}}} [\![E\sigma_3]\!].$$

Theorem 3 can be concluded from the previous results as follows.

Proof:

If $E_1 \cong E_2$ then there exists injective renamings $\sigma_1 : \mathbf{SKeys} \to \mathbf{SKeys}, \sigma_2 : \mathbf{Nonce} \to \mathbf{Nonce}$, and ldp-renaming $\sigma_3 : \mathsf{poly}(E_2) \to \mathbf{Poly}$ such that $\mathsf{pattern}(E_1) = \mathsf{pattern}(E_2\sigma_1\sigma_2\sigma_3)$. Therefore we have the following sequence of equivalences:

$$[\![E_2]\!] \approx_{\epsilon_1, t_1} [\![\mathsf{pattern}(E_2)]\!] \approx_{\epsilon_2, t_2} [\![\mathsf{pattern}(E_2\sigma_1\sigma_2\sigma_3)]\!] = [\![\mathsf{pattern}(E_1)]\!] \approx_{\epsilon_3, t_3} [\![E_1]\!].$$

where the first and last equivalences follow from Theorem 2, the second equivalence follows by Lemma 6, and the third equality from the assumptions that E_1 and E_2 are symbolically equivalent. Notice that all of the parameters involved depend on the underlying group $\mathbb{G}$ and also on the particular expressions E_1 and E_2. □

4.5. An example

To appreciate the power that the above soundness theorem provides, consider the expression:

$$E(F) = \left(\mathsf{g}^{X_1}, \mathsf{g}^{X_2}, \mathsf{g}^{X_3}, \mathsf{g}^{X_1X_2}, \mathsf{g}^{X_1X_3}, \mathsf{g}^{X_2X_3}, \{\!|K|\!\}^{\mathsf{s}}_{h(\mathsf{g}^{X_1X_2X_3})}, \{\!|F|\!\}^{\mathsf{s}}_{K}\right)$$

where F is some arbitrary expression. Expression E represents the transcript of the executions of the following (toy) protocol: three parties with secret keys X_1, X_2 and X_3 first agree on a common secret key $h(\mathsf{g}^{X_1X_2X_3})$ (by broadcasting the first 6 messages in the expression). Then, one of the parties generates a new key K which it broadcasts to the other parties encrypted under $h(\mathsf{g}^{X_1X_2X_3})$. Finally, one of the parties, sends some secret expression F encrypted under K. To argue about the security of the secret expression (against a passive adversary) it is sufficient to show that the distributions associated to the expressions $E(F)$ and $E(0)$ are indistinguishable.

Although conceptually simple, a full cryptographic proof would require several reductions (to DDH and security of encryption), and most likely would involve at least

one hybrid argument (for proving the security of encrypting K under $h(g^{X_1X_2X_3})$). The tedious details of such a proof can be entirely avoided by using the soundness theorem: it is straightforward to verify that $E(F) \cong E(0)$, and in general, this procedure could potentially be automated. Since $E(F)$ is acyclic, the desired result follows immediately by Theorem 3.

References

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS 2000*, pp. 3–22.

[2] A. Datta, A. Derek, J.C.Mitchell, V.Shmatikov, M. Turuani. Probabilistic polynomial-time semantics for a Protocol Security Logic. In *ICALP 2005* , pp. 16–29.

[3] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *CCS 2003*, pp. 220–230.

[4] F. Bao, R. Deng, and H. Zhu. Variations of Diffie-Hellman problem. In *ICICS 2003*, pp. 301–312.

[5] G. Barthe, B. Gregoire, R. Janvier, and S. Zanella Béquelin. A framework for language-based cryptography proofs. In *2nd Informal ACM SIGPLAN Workshop on Mechanizing Metatheory*, 2007.

[6] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: security, proofs, and improvements. In *EUROCRYPT 2000*, pp. 259–274.

[7] M. Bellare and P. Rogaway. The game-playing technique and the security of triple encryption. in *CRYPTO 2006*, pp. 409–426.

[8] M. Bellare, A. Desai, E. Jokipii, and Ph. Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. In *FOCS 1997*, pages 394–403.

[9] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. of Computing*, 13:850–864, 1984.

[10] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT 2005*, pp. 440–456.

[11] E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. In *ASIACRYPT 2002*, pp. 497–514.

[12] E. Bresson, O. Chevassut, and D. Pointcheval. The group Diffie-Hellman problems. In *SAC 2002*, pp. 325–338.

[13] E. Bresson, Y. Lakhnech, L. Mazarè, and B. Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In *Crypto 2007*, pp. 482–499.

[14] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system (extended abstract). In *EUROCRYPT 1994*, pp. 275–286.

[15] B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In *CRYPTO 2006*, pp. 537–554.

[16] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145.

[17] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *CRYPTO 1997*, pp. 455–469.

[18] Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FSTTCS 2003*, pp. 124–135.

[19] D. Coppersmith and I. Shparlinski. On polynomial approximation of the discrete logarithm and the Diffie-Hellman mapping. *J. of Cryptology*, 13(2):339–360, 2000.

[20] V. Cortier, S. Kremer and B. Warinschi. A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems. *J. of Automated Reasoning*.

[21] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO 1998*, pp. 13–25.

[22] D. Dolev and A. Yao. On the security of public key protocols. *IEEE IT*, 29(12):198–208, 1983.

[23] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE IT*, 31(4):469–472, 1985.

[24] O. Goldreich. Foundations of Cryptography – Basic Tools. *Cambridge University Press*, 2004.

[25] J. Couran, M. Dabugnard, C. Ene, P. Lafourcade, and Y. Lakhnech. Towards Automated Proofs for Asymmetric Encryption in the Random Oracle Model. In *ACM CCS 2008*, pp. 371-380, 2008.
[26] U. Maurer and S. Wolf. Diffie-Hellman oracles. In *CRYPTO 1996*, pp. 268–282.
[27] D. Micciancio and S. Panjwani. Adaptive security of symbolic encryption. In *TCC 2005*, pp. 245–263.
[28] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC'04*, pp. 133–151.
[29] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *J. of Computer Security*, 12(1):99–129, 2004. Preliminary version in WITS 2002.
[30] J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *CSFW 2003*, pp. 47–61.
[31] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS 1997*, pp. 458–467.
[32] M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *J. of Theoretical Computer Science*, 1-3(299):451–475, 2003.
[33] A.-R. Sadeghi and M. Steiner. Assumptions related to discrete logarithms: Why subtleties make a real difference. In *EUROCRYPT 2001*, pp. 244–261.
[34] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT 1997*, pp. 256–266.
[35] V. Shoup. Sequences of games: a tool for taming complexity of security proofs. Cryptology ePrint Archive 2004/332.
[36] I. Shparlinski. Security of most significant bits of g^{x^2}. *IPL*, 83(2):109–113, 2002.
[37] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *ACM CCS 1996*, pp. 31–37.

A. Proof of Lemma 3

Recall that here we show that if E is an expression in which $K_0 \in \mathbf{SKeys}$ does not occur encrypted in E (it only occurs as an encryption key), and if Π is an (ϵ_Π, t_Π)-$\mathsf{IND\text{-}CPA}^*$ secure encryption scheme then replacing terms encrypted under K_0 with $\Box$ does not change the distribution of the expression significantly. Formally, we show that: $[\![E]\!]_{\Pi,\mathbb{G}} \approx_{\epsilon,t} [\![E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]]\!]_{\Pi,\mathbb{G}}$ for some $\epsilon = \epsilon_\Pi$ and $t = t_\Pi - t_{sim\mathcal{B}}$ for some constant $t_{sim\mathcal{B}}$ that we determine during the proof. The intuition is that a distinguisher that observes the change in the distribution must observe the change of plaintext under a key he dose not know, and hence breaks encryption.

Given a distinguisher for distributions $[\![E]\!]$ and $[\![E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]]$ we construct an adversary $\mathcal{B}$ against Π. The construction is in Figure 6. The idea of the construction is to generate a bitstring e such that if the oracle to which $\mathcal{B}$ has access is the real oracle (i.e. it returns true encryptions of the message that is queried) then e is sampled according to $[\![E]\!]$, and if the oracle of $\mathcal{B}$ is the fake oracle (which returns encryptions of 0) then the e is sampled according to $[\![E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{K_0}]]$. To accomplish this task, $\mathcal{B}$ executes the $\mathsf{Init}(E)$ algorithm (given earlier in Figure 4) to obtain bitstring interpretation τ for all symbols that occur in E. Then, $\mathcal{B}$ uses τ to produce bitstring interpretations for all of the terms that occur in E much in the same way as the algorithm Gen of Figure 4. The only difference is that encryptions under K_0 are obtained via the oracle to which $\mathcal{B}$ has access (line 5 of the modified generation algorithm mGen.) The sample e that is generated satisfies the desired property. If the oracle to which $\mathcal{B}$ has access is a real oracle $\mathcal{E}_k(\cdot)$, symbolic encryptions under K_0 are interpreted as encryptions under k. In other words the result of $\mathsf{mGen}(E, K_0, \tau)$ is the same as the result of $\mathsf{Gen}(E, \tau')$ where τ' is τ except that the key assigned to K_0 is set to be k, the key of the encryption oracle. In the case when the oracle of $\mathcal{B}$ is a fake oracle, the oracle that returns $\mathcal{E}_k(0)$ each time

Algorithm $\mathsf{mGen}^{\mathcal{O}}(E, K_0, \tau)$
If $E \in \mathbf{SKeys} \cup \mathbf{Nonce} \cup \{\Box\}$ then $e \leftarrow \tau(E)$
If $E = (E_1, E_2)$ then $e_1 \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $e_2 \xleftarrow{\$} \mathsf{Gen}(E_2, \tau)$; $e \leftarrow \langle e_1, e_2 \rangle$
If $E = \mathsf{g}^{p(X_1, X_2, \ldots, X_n)}$ then $e \leftarrow \mathsf{g}^{p(\tau(X_1), \tau(X_2), \ldots, \tau(X_n))}$
If $E = \{\!|E_1|\!\}^{\mathsf{s}}_K$ and $K \neq K_0$ then $m \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $e \xleftarrow{\$} \mathcal{E}(\tau(K), m)$;
If $E = \{\!|E_1|\!\}^{\mathsf{s}}_K$ and $K = K_0$ then $m \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $e \xleftarrow{\$} \mathcal{O}(m)$;
If $E = \{\!|E_1|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)}$ then $m \xleftarrow{\$} \mathsf{Gen}(E_1, \tau)$; $k \leftarrow \mathsf{Gen}(\mathsf{g}^p)$; $e \leftarrow \mathcal{E}(\mathsf{kd}(k), m)$
Return e
Adversary $\mathcal{B}^{\mathcal{O}}(E, K_0)$
$\tau \xleftarrow{\$} \mathsf{Init}(E)$
$e \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}}(E, K_0, \tau)$
output $\mathcal{D}(e)$

Figure 6. Adversary $\mathcal{B}$ is against encryption. It uses procedure mGen which makes use of the oracle of $\mathcal{B}$.

it is queried, then all encryptions under K_0 are interpreted as encryptions under k of 0. This means that e produced by mGen is sampled from $E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]$. Notice that in both cases it is crucial that K_0 occurs in E only in key positions and not as messages. Otherwise $\mathcal{B}$, who has access to k (the interpretation of K_0) only indirectly through its oracle, would not be able to produce a consistent interpretation for E. We therefore have that:

$$
\begin{aligned}
&\mathbf{Adv}^{\mathsf{ind\text{-}cpa}^*}_{\Pi}(\mathcal{B}) = \\
&= \left| \Pr\left[\mathcal{B}^{\mathcal{E}_k(\cdot)} = 1\right] - \Pr\left[\mathcal{B}^{\mathcal{E}_k(0)} = 1\right] \right| \\
&\geq \left| \Pr\left[x \xleftarrow{\$} [\![E]\!] : \mathcal{D}(x) = 1\right] - \Pr\left[x \xleftarrow{\$} [\![E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}] : \mathcal{D}(x) = 1]\!]\right] \right|
\end{aligned}
$$

The right hand-side of the last inequality is exactly the advantage of $\mathcal{D}$ in distinguishing distributions $[\![E]\!]$ and $E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]$. If t_D is the running time of $\mathcal{D}$ then the running time of $\mathcal{B}$ is $t_{\mathcal{B}} = t_{sim\mathcal{B}} + t_{\mathcal{D}}$, where $t_{sim\mathcal{B}}$ is some constant time that $\mathcal{B}$ spends in preparing the sample e. If Π is (ϵ_Π, t_Π)-IND-CPA* then we immediately obtain that $[\![E]\!] \approx_{t,\epsilon} [\![E[\{\!|\ |\!\}^{\mathsf{s}}_{K_0} \mapsto \{\!|\Box|\!\}^{\mathsf{s}}_{K_0}]]\!]$, where $t = t_\Pi - t_{sim\mathcal{B}}$ and $\epsilon = \epsilon_\Pi$.

B. Proof of Lemma 4

Recall that here we show that if E is an acyclic expression in which $h(\mathsf{g}^{p_0})$ only occurs as an encryption key then one can replace each occurrence of $h(\mathsf{g}^{p_0})$ in E with a new key K_0 without significantly changing the distribution of the expression. More precisely, we show how to turn a distinguisher $\mathcal{D}$ between distributions $[\![E]\!]$ and $[\![E[\{\!|\ |\!\}^{\mathsf{s}}_{h(\mathsf{g}^{p_0})} \mapsto \{\!|\ |\!\}^{\mathsf{s}}_{K_0}]]\!]$ into an attacker for the $(\mathsf{poly}(E) \setminus \{p\}, \{p\})$-DDH assumption. To simplify notation (and in line with the notation of Section 3) in the remainder of this section we write P for the set $\mathsf{poly}(E) \setminus \{p_0\}$ and Q for the set $\{p_0\}$.

The adversary $\mathcal{C}$ that we construct is in Figure 7.

Recall that $\mathcal{C}$ has access to two oracles. The first oracle accepts as queries polynomials $p \in P$, and returns $\mathsf{g}^{p(x_1, x_2, \ldots, x_n)}$. The second oracle accepts a single query, p_0

Algorithm $\mathsf{mGen}^{\mathcal{O}_1,\mathcal{O}_2}(E, p_0, \tau)$
If $E \in \mathbf{SKeys} \cup \mathbf{Nonce} \cup \{\Box\}$ then $e \leftarrow \tau(E)$
If $E = (E_1, E_2)$ then $e_1 \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}_1,\mathcal{O}_2}(E_1, \tau)$; $e_2 \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}_1,\mathcal{O}_2}(E_2, \tau)$; $e \leftarrow \langle e_1, e_2 \rangle$
If $E = \mathsf{g}^{p(X_1,X_2,\ldots,X_n)}$ then $e \xleftarrow{\$} \mathcal{O}_1(p)$
If $E = \{E_1\}_K$ then $m \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}_1,\mathcal{O}_2}(E_1, \tau)$; $e \xleftarrow{\$} \mathcal{E}(\tau(K), m)$;
If $E = \{E_1\}_{h(\mathsf{g}^p)}$ and $p \neq p_0$ then $k \xleftarrow{\$} \mathcal{O}_1(p)$; $m \xleftarrow{\$} \mathsf{mGen}(E_1, \tau)$; $e \leftarrow \mathcal{E}(k, m)$
If $E = \{E_1\}_{h(\mathsf{g}^p)}$ and $p = p_0$ then $k \xleftarrow{\$} \mathcal{O}_2(p)$; $m \xleftarrow{\$} \mathsf{mGen}(E_1, \tau)$; $e \leftarrow \mathcal{E}(\mathsf{kd}(k), m)$
Return e
Adversary $\mathcal{C}^{\mathcal{O}_1,\mathcal{O}_2}(E, p_0)$
$\tau \xleftarrow{\$} \mathsf{Init}(E)$
$e \xleftarrow{\$} \mathsf{mGen}^{\mathcal{O}_1,\mathcal{O}_2}(E, p_0, \tau)$
output $\mathcal{D}(e)$

Figure 7. Adversary $\mathcal{C}$ is against the $(\mathsf{poly}(E) \setminus \{p_0\}, \{p_0\})$-DDH assumption. It uses procedure mGen which makes use of the oracle of $\mathcal{B}$.

and behaves in one of two ways: in the first case it returns $\mathfrak{g}^{p_0(x_1,x_2,\ldots,x_n)}$ in the second case it returns $\mathfrak{g}^{r_1}$. The task of the adversary is to determine in which of the two possible situations it is.

Starting from E, adversary $\mathcal{C}$ constructs a sample e. First it executes the $\mathsf{Init}(E)$ algorithm to obtain an interpretation for all of the symbols that occur in E and then recursively constructs e. The construction of e is similar to the way the generation algorithm Gen works, except that the interpretation of group elements that occur in E is obtained from the oracles of $\mathcal{C}$. For all polynomials p in P with $p \neq p_0$, the generation procedure obtains the interpretation of g^p from oracle $\mathcal{O}_1$ (which returns the true value of g^p under the interpretation for variables that it maintains internally). The interpretation of g^{p_0} is obtained from its second oracle. We distinguish two cases. When $\mathcal{C}$ interacts with the "real" oracle it obtains back the interpretation of g^{p_0}. Therefore, in this case, e is distributed according to $[\![E]\!]$. In the fake game the value returned is g^r for a random exponent r. Since we assumed that the distribution of $\mathcal{KG}$ and that of $\mathsf{kd}(\mathsf{g}^r)$ are identical, then symbolic encryptions under $h(\mathsf{g}^{p_0})$ are interpreted as encryptions under a random key. It follows that in this case e is distributed according to $[\![E[\{\!|\,|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\,|\!\}^{\mathsf{s}}_{K_0}]]\!]$ for some K_0 that does not occur in E. We therefore have that:

$$
\begin{aligned}
&\mathbf{Adv}^{(P,Q)\text{-}\mathsf{ddh}}_{\mathbb{G}}(\mathcal{C}) = \\
&= \left|\Pr\left[\mathcal{C}^{\mathsf{Real}_{(P,Q)}} = 1\right] - \Pr\left[\mathcal{C}^{\mathsf{Fake}_{(P,Q)}} = 1\right]\right| \\
&= \left|\Pr\left[e \xleftarrow{\$} [\![E]\!] : \mathcal{D}(e) = 1\right] - \Pr\left[e \xleftarrow{\$} [\![E[\{\!|\,|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\,|\!\}^{\mathsf{s}}_{K_0}]]\!] : \mathcal{D}(e) = 1\right]\right|
\end{aligned}
$$

Notice that the last term is the advantage that $\mathcal{D}$ has in distinguishing between distributions $[\![E]\!]$ and $[\![E[\{\!|\,|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\,|\!\}^{\mathsf{s}}_{K_0}]]\!]$. If the running time of $\mathcal{D}$ is $t_{\mathcal{D}}$ then the running time of $\mathcal{C}$ is $t_{\mathcal{D}} + t_{sim\mathcal{C}}$. Here $t_{sim\mathcal{C}}$ is the time that $\mathcal{C}$ uses to compute the sample e. If $\mathbb{G}$ is $(\epsilon_{\mathbb{G}}, t_{\mathbb{G}})$-appropriate for E, then it follows that $[\![E]\!] \approx_{\epsilon,t} [\![E[\{\!|\,|\!\}^{\mathsf{s}}_{h(\mathsf{g}^p)} \mapsto \{\!|\,|\!\}^{\mathsf{s}}_{K_0}]]\!]$ for $\epsilon = \epsilon_{\mathbb{G}}$ and $t = t_{\mathbb{G}} - t_{sim\mathcal{C}}$.

□