Computational Soundness for Standard Assumptions of Formal Cryptography

by

Jonathan Herzog

B.S., Harvey Mudd College (1997))
M.S., Massachusetts Institute of Technology (2002)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author . . . .
Department of Electrical Engineering and Computer Science
May 24, 2004

Certified by . . . .
Ron Rivest
Professor, MIT
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Computational Soundness for Standard Assumptions of Formal Cryptography

by

Jonathan Herzog

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The *Dolev–Yao* model is a useful and well-known framework in which to analyze security protocols. However, it models the messages of the protocol at a very high level and makes extremely strong assumptions about the power of the adversary. The *computational* model of cryptography, on the other hand, takes a much lower-level view of messages and uses much weaker assumptions.

Despite the large differences between these two models, we have been able to show that there exists a relationship between them. Previous results of ours demonstrate that certain kinds of computational cryptography can result in an equivalence of sorts between the formal and computational adversary. Specifically:

- We gave an interpretation to the messages of the Dolev–Yao model in terms of computational cryptography,
- We defined a computational security condition, called *weak Dolev-Yao non-malleability*, that translates the main assumptions of the Dolev-Yao model into the computational setting, and
- We demonstrated that this condition is satisfied by a standard definition of computational encryption security called *plaintext awareness*.

In this work, we consider this result and strengthen it in four ways:

1. Firstly, we propose a stronger definition of Dolev-Yao non-malleability which ensures security against a more adaptive adversary.
2. Secondly, the definition of plaintext awareness is considered suspect because it relies on a trusted third party called the *random oracle*. Thus, we show that our new notion of Dolev-Yao non-malleability is satisfied by a weaker and less troublesome definition for computational encryption called *chosen-ciphertext security*.
3. Thirdly, we propose a new definition of plaintext-awareness that does not use random oracles, and an implementation. This implementation is conceptually simple, and relies only on general assumptions. Specifically, it can be thought of as a 'self-referential' variation on a well-known encryption scheme.
4. Lastly, we show how the computational soundness of the Dolev-Yao model can be maintained even as it is extended to include new operators. In particular, we show how the Diffie-Hellman key-agreement scheme and the computational Diffie-Hellman assumption can be added to the Dolev-Yao model in a computationally sound way.

Thesis Supervisor: Ron Rivest
Title: Professor, MIT

Contents

Chapter 1

Introduction

The area of *formal cryptography* has had two beginnings: the first came in 1978, when Needham and Schroeder proposed the first set of authentication protocols [53]. The second beginning came seventeen years later, when Gavin Lowe found a flaw in Needham and Schroeder's public key protocol, fixed the protocol, and—most importantly—proved that the fixed protocol was correct [38, 39]. What distinguished these two beginnings from previous cryptographic efforts was the level of abstraction: the Needham and Schroeder protocols were specified in terms of abstract cryptographic operations rather than specific cryptographic algorithms. Likewise, the flaw found by Lowe did not rely upon the properties of the cryptographic algorithms, and existed even in the abstracted system.

What are authentication protocols? There is no exact definition, but the examples share many characteristics:

1. The protocols are sequences of messages between two or three parties,
2. The messages utilize cryptographic algorithms to "secure" the contents in various ways,
3. The protocol as a whole is intended to perform one or both of two objectives:
    - Authentication: over the course of the protocol, one of the parties should gain proof that another particular party is also participating in the protocol, that they share common views of the protocol, and that they agree on the values used in the protocol.
    - Secrecy: Some or all of the values agreed upon during the protocol should be unknown to other observers.

Several real-world protocols conform to this definition: SSH [64], TLS [18], and KERBEROS [33] are the three most widely-known. Being real-world protocols, however, they are quite large and complex. We will use a much simpler protocol from the literature to illustrate the definition.

1.1 An Example Protocol

The Needham–Schroeder public key protocol can be described as a sequence of three messages between two parties:

1. $A \to B : \{\!|A\, N_1|\!\}_{K_B}$
2. $B \to A : \{\!|N_1\, N_2|\!\}_{K_A}$
3. $A \to B : \{\!|N_2|\!\}_{K_B}$

In this notation, A and B are names or unique identifiers of two parties. Principal A starts the protocol, and hence has the role of *initiator*. Principal B responds to A's original message, and hence has the role of *responder*.[1] A begins the protocol by sending the message $\{\!|A\, N_1|\!\}_{K_B}$ to B, where

- $\{\!|M|\!\}_K$ is an encryption of the plaintext M with the key K, and
- $M_1\, M_2$ is the concatenation, or pairing, of messages M_1 and M_2.

The value N_1 is a *nonce*, a random value generated freshly by (in this case) A, and of sufficient length to make infinitesimally small the chances of its previous use by any other party. N_2 is a nonce generated by B, which it includes in the second message.

The protocol assumes the existence of a (collision-free) mapping from names to public keys; K_i is the public key belonging to the name i.[2] Identity is defined by knowledge of secret keys: Entity i is defined to be anyone who knows the secret key associated with K_i. When interpreting these protocols, however, it is usually assumed that secret keys are known by one—and only one—entity, and never shared or discovered through cryptanalysis.

1.2 Security Goals and Flaws

The Needham–Schroeder public key protocol has both secrecy and authentication goals:

1. Secrecy: The two values N_1 and N_2 should only be known to A and B. More generally, the nonces used in the protocol should only be known to the two participants, and
2. Authentication: The initiator and responder should be authenticated to each other. Specifically:

[1] Though it is conceivable to have a protocol without analogous roles, they are always found in practice. Also, it is not unusual for the initiator and responder to use the services of a trusted third party called the *server*.

[2] In practice, this mapping could be instantiated by a PKI or similar mechanism. The situation for secret keys is similar—a partial mapping from sets of names to keys is assumed—but the analysis typically is more complex.

- The initiator should know the identity of the responder (i.e., that the responder knows the appropriate secret key) that the responder knows who the initiator is, and that they agree on the values of the nonces used, and
- The responder should know the identity of the initiator, that the initiator knows who the responder is, and that they agree on the values of the nonces used.

However, the authentication conditions do not hold. In the attack that Lowe discovered, the initiator (A) starts a run of the protocol with a malicious entity (M), who then pretends to be the initiator ($M(A)$) to a third party (B):

1. $A \rightarrow M : \{\!|A, N_1|\!\}_{K_M}$
2. $M(A) \rightarrow B : \{\!|A, N_1|\!\}_{K_B}$
3. $B \rightarrow M(A) : \{\!|N_1, N_2|\!\}_{K_A}$
4. $M \rightarrow A : \{\!|N_1, N_2|\!\}_{K_A}$
5. $A \rightarrow M : \{\!|N_2|\!\}_{K_M}$
6. $M(A) \rightarrow B : \{\!|N_2|\!\}_{K_B}$

The entity B is fooled by the above attack. From B's perspective, the sequence of messages is exactly what it would expect from a run with the initiator A. While the initiator A is engaged in a run of the protocol, it thinks that the responder is not B but M. If, for example, A is an individual, M is an on-line merchant, and B is A's bank, then when A runs the protocol with M in order to place an order, M can masquerade as A to her bank to empty her account.

Lowe's fix is to include the responder's name in the second message, making it:

2. $B \rightarrow A : \{\!|B\, N_1\, N_2|\!\}_{K_A}$

With this simple change, he proves that the protocol satisfies the security goals—assuming that the adversary does not break or otherwise manipulate encryptions (aside from decrypting them with the appropriate key). The proof of correctness involves two parts: he first proves that any attack on a large system (i.e., multiple honest parties) could be translated into an attack on the small system of just two parties. He then used a model checker, a standard tool in the formal methods community, to exhaustively search the small system for vulnerabilities.

Since Lowe's first two papers, there has been a flurry of interest in the area of *formal cryptography*, as this field came to be known. The use of model checkers has been expanded and refined [50], theorem provers have been shown to be of value [55], and more direct mathematical methods have proved to be quite useful [63]. (See [43] for a recent survey of the field.) However, all these methods share common characteristics, and hence common weaknesses. They all use methods use the Dolev-Yao model [20], an extremely abstract and high-level framework that bears further and careful examination.

1.3 The Dolev–Yao Model

The Dolev-Yao model is an early and successful mathematical framework in which to examine cryptographic protocols.[3] In this model, messages are assumed to be elements of an algebra $\mathcal{A}$ of values. There are two types of atomic messages:

- Texts ($\mathcal{T}$) with two sub-types: identifiers (public, predictable, denoted by $\mathcal{M}$) and random nonces (private, unpredictable, denoted by $\mathcal{R}$), and
- Keys ($\mathcal{K}$) with two sub-types: public keys ($\mathcal{K}_{Pub}$) and private keys ($\mathcal{K}_{Priv}$)

Compound messages are created by two deterministic operations:

- $\mathit{encrypt} : \mathcal{K}_{Pub} \times \mathcal{A} \rightarrow \mathcal{A}$
- $\mathit{pair} : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$

We write $\{\!|M|\!\}_K$ for $\mathit{enc}(K, M)$ and $M\,N$ for $\mathit{pair}(M, N)$.[4] We require that there be a bijection

$$\mathit{inv} : \mathcal{K}_{Pub} \rightarrow \mathcal{K}_{Priv}$$

and by K^{-1} we mean $\mathit{inv}(K)$ when K is a public key and $\mathit{inv}^{-1}(K)$ when K is a private key.

The model places a strong assumption on the set of messages:

Assumption 1 *The algebra $\mathcal{A}$ is* free*: every value has a unique representation.*

That is, messages can be thought of as *being* parse trees. This implies, among other things, that there is no other way to produce the encryption $\{\!|M|\!\}_K$ than to encrypt the message M with the key K. (We discuss this and similar implications of this assumption in more depth later.)

There are two kinds of active parties in this model: honest participants and the adversary. The honest participants follow the steps of the protocol without deviation. They can engage in multiple runs of the protocol simultaneously and with different parties. However, the network is assumed to be completely under the control of the adversary, who can record, delete, replay, reroute, and

[3] There are actually several variations on the Dolev-Yao model, each tailored to a specific tool or application. We provide and discuss a generic example that uses only public-key encryption. There are other versions that use both symmetric and asymmetric encryption, or simply symmetric, but we do not consider them here.

[4] When three or more terms are written together, such as $M_1\,M_2\,M_3$, we assume they are grouped to the left. That is, $M_1\,M_2\,M_3 = \mathit{pair}(\mathit{pair}(M_1, M_2), M_3)$.

reorder messages. This is modeled by letting the adversary be the "central exchange" of the model: all messages are sent either from honest participant to adversary or vice-versa.

Assumption 2 *Each execution of a protocol in the Dolev-Yao model is an alternating sequence of messages ("queries," $q_i \in \mathcal{A}$) from the adversary and messages ("responses," $r_i \subseteq \mathcal{A}$) from honest principals:*

$$r_0 \quad q_1 \quad r_1 \quad q_2 \quad r_2 \quad \ldots \quad q_{n-1} \quad r_{n-1} \quad q_n \quad r_n$$

(Here, r_0 is a setting-specific adversary "initialization" that represents any additional information the adversary could have learned due to the setting and environment.)

The sequence usually contains more information than merely the message contents, such as the intended or actual recipient, but we will ignore these extra details in this work.

The model makes strong assumptions regarding the adversary. In particular, it is assumed that every query q_i is *derivable* from the adversary's initial knowledge and r_0, r_1, $r_2 \ldots r_{i-1}$. The initial knowledge of the adversary includes at least the following:

- the public keys ($\mathcal{K}_{Pub}$),
- the private keys of subverted participants ($\mathcal{K}_{Adv} \subseteq \mathcal{K}_{Priv}$),
- the identifiers of the principals ($\mathcal{M}$), and
- the nonces it itself generates ($\mathcal{R}_{Adv}$) which are assumed to be distinct from all nonces generated by honest participants.

For a given message M to be derivable from a set of messages S it must be possible to produce it by applying the following operations a finite number of times:

- decryption with known or learned private keys,
- encryption with public keys,
- pairing of two known elements, and
- separation of a "join" element into its component elements.

To combine these lists of adversary knowledge and abilities:

Definition 1 (Closure) *The* closure of S*, written* $C[S]$*, is the smallest subset of* $\mathcal{A}$ *such that:*

1. $S \subseteq C[S]$,
2. $\mathcal{M} \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Adv} \cup \mathcal{R}_{Adv} \subseteq C[S]$,
3. *If* $\{\!|M|\!\}_K \in C[S]$ *and* $K^{-1} \in C[S]$*, then* $M \in C[S]$,

4. *If $M \in C[S]$ and $K \in C[S]$, then $\{\!|M|\!\}_K \in C[S]$,*

5. *If $M\,N \in C[S]$, then $M \in C[S]$ and $N \in C[S]$, and*

6. *If $M \in C[S]$ and $N \in C[S]$, then $M\,N \in C[S]$.*

It is the central assumption of the Dolev-Yao model that this closure operation represents the limit of the ability of the adversary to create new messages:

Assumption 3 *If the Dolev-Yao adversary knows a set S of messages, it can produce only messages in $C[S]$.*

Hence, in the Dolev-Yao model, it must be that $q_i \in C[\{r_0, r_1, \ldots r_{i-1}\}]$.

The model also makes an assumption regarding the honest participants, but it is quite weak and applies equally to the adversary:

Assumption 4 *During a protocol run, the participants (including the adversary) will see only messages in the algebra $\mathcal{A}$ and will respond only with messages in $\mathcal{A}$.*

The three assumptions of the Dolev-Yao model—the freeness of the algebra, the limitations on the adversary, and the restriction of messages to the algebra—are also its weaknesses. It is clear that these assumptions greatly simplify the task of protocol analysis. The techniques of the Dolev-Yao model are easy to apply, and the task of protocol analysis in this model has been automated [61]. It is not clear, however, that these assumptions are at all justified, and this casts doubt onto any method that uses them. It is true that many attacks and flaws have been found even in the presence of these assumptions. However, if a method based on these high-level assumptions finds no flaws in a protocol, there still remains the possibility that some attack can be found when the adversary has an additional, reasonable, ability.

1.4 The Computational Model

The assumptions of the Dolev–Yao model seem especially strong when compared to those made by alternate models. The other widely-accepted model of encryption and cryptographic algorithms is that of the field of *computational* cryptography, which is much lower-level and much less abstract than the Dolev-Yao model. It regards cryptography as a branch of complexity theory, and regards cryptographic primitives as algorithms that map bit-strings into bit-strings. Adversaries are regarded as efficient algorithms, and security conditions are stated in terms of the probability that an adversary can perform a given calculation in the face of an increasing "security parameter."

For example, in this framework encryption is not an abstract operation but a collection of efficient algorithms. "Efficient" in this setting means that the algorithm runs in probabilistic polynomial time (PPT): polynomial-time in the security parameter and with access to a source of random bits.

Here, the security parameter indicates the amount of time the key-generation algorithm can take to produce a new (and random) key, and so corresponds roughly to the size of keys.

Definition 2 (Public-Key Encryption) *A public-key encryption scheme is a triple of algorithms* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$[5]*:*

- $\mathsf{G} : \mathsf{Parameter} \rightarrow \mathsf{PublicKey} \times \mathsf{PrivateKey}$ *is the (randomized) key generation algorithm,*
- $\mathsf{E} : \mathsf{String} \times \mathsf{PublicKey} \rightarrow \mathsf{Ciphertext}$ *is the (randomized) encryption algorithm, and*
- $\mathsf{D} : \mathsf{String} \times \mathsf{PrivateKey} \rightarrow \mathsf{String} \cup \{\bot\}$ *is the decryption algorithm, which we assume returns* $\bot$ *whenever the input string is not a valid encryption under the corresponding public key.*

It is required that for all values of the security parameter $\eta \in \mathsf{Parameter}$*, all messages* m*, and all key-pairs* $(\mathsf{e}, \mathsf{d}) \leftarrow \mathsf{G}(1^\eta)$*,* $\mathsf{D}(\mathsf{E}(m, \mathsf{e}), \mathsf{d}) = m$*.*

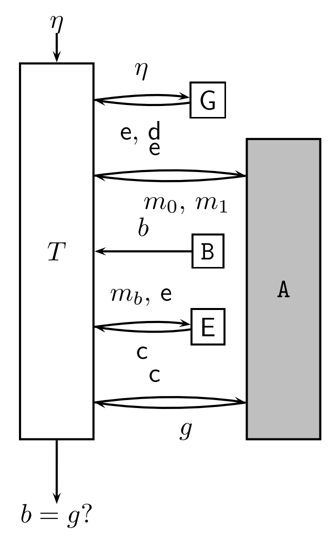
Note that the key-generation algorithm is randomized, as is the encryption algorithm. Hence, the key-generation algorithm produces a probability distribution on keys, and the encryption algorithm produces a probability distribution on ciphertexts for each message/key pair. Also note that security parameter η is given to the key generation algorithm G in unary (in the form 1^η) so that G can run in time polynomial in η rather than in $\log(\eta)$ (the number of bits required to represent the value of η). Without loss of generality, we assume that for a given value of the security parameter η, the key-generation and encryption algorithms use η bits of randomness.

Although the adversary of the computational model is much less limited than its Dolev-Yao counterpart, it is not all-powerful. It can be any arbitrary algorithm, so long as it is efficient (PPT). This allows the methods of computational cryptography constrain the adversary, typically by proving a *reduction* from an encryption scheme to some other problem. This reduction shows that if any adversary can break the security of the encryption scheme, it can be modified to efficiently solve some other particular problem. In particular, encryption schemes can be reduced to problems known (or, as is more commonly the case, widely believed) to be harder than PPT. Thus, an encryption scheme that reduces to these problems cannot be (or is unlikely to be) broken by any efficient adversary.

The most basic notion of security for a public-key encryption scheme is that of *semantic security.* In essence, this definition of security states that the adversary should be unable to distinguish an encryption of message m_0 from an encryption of m_1, even if the adversary can choose m_0 and m_1 based on the public key. More explicitly, the definition of semantic security concerns a game

[5] *The helper sets are:*

- $\mathsf{Parameter} = \mathcal{N}$*,*
- $\mathsf{PublicKey}$*,* $\mathsf{PrivateKey}$ *and* $\mathsf{Ciphertext}$ *vary between key generation algorithms and implicitly depend on the parameter, and*
- $\mathsf{String} = \{0, 1\}^*$



(Time flows from top to bottom.)

Figure 1-1: The semantic security "game"

between the adversary and a "tester" that consists of a series of "experiments." This game is shown pictorially in Figure 1-1 (where the adversary is shaded to emphasize that it alone can be an arbitrary algorithm).

- First, the tester receives the security parameter η.
- The tester then honestly and randomly generates a key pair (e, d) according to the security parameter it was given and the key generation algorithm G.
- The adversary is given the public encryption key e by the tester and allowed to perform any computation it likes.
- The adversary then gives the tester any two messages it likes (m_0 and m_1) of the same length. At this point, it then goes dormant but does not terminate or erase its state.
- The tester flips a coin to produce a bit b, which will determine which of m_0 and m_1 will be encrypted.
- The chosen message m_b is encrypted to produce a ciphertext c.
- The tester gives to the adversary A all of the public information to date: the public key e, the two messages m_0 and m_1, and the ciphertext c.The adversary then resumes computation on the new input, but begins from the internal state it had when it went dormant.[6] It then can

[6]The equivalent, "standard" form of this definition has the adversary begin from the same initial state each time, but the first invocation produces some state information s which is given to the second invocation. However, we will adopt as a convention for this paper that the adversary keeps state between multiple invocations.

perform any efficient computation it likes, but must produce a “guess” g.

The adversary “wins” the game if it managed to guess the value of b (i.e., if $b = g$). Clearly, a simple adversary that makes a random guess can win half the time. The definition of semantic security requires that the *advantage* of the adversary—the degree to which the adversary can do better than a random guess—must become vanishingly small very fast. In fact, the adversary’s advantage must shrink faster than any polynomial.

In the standard notation (which treats the tester as implicit):

Definition 3 (Semantic Security) *A public-key encryption algorithm* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is* (one-pass) semantically secure[7] *if:*

$$\forall \textit{ PPT algorithms } \mathtt{M} \textit{ and } \mathtt{A}, \forall \textit{ polynomials } q, \forall \textit{ sufficiently large } \eta,$$

$$\begin{aligned}\Pr[\quad & (\mathsf{e}, \mathsf{d}) \leftarrow \mathsf{G}(1^\eta);\\ & m_0, m_1, s \leftarrow \mathtt{M}(1^\eta, \mathsf{e});\\ & b \leftarrow \{0,1\}\,;\\ & \mathsf{c} \leftarrow \mathsf{E}_{\mathsf{e}}(m_b) :\\ & \mathtt{A}(1^\eta, \mathsf{e}, m_0, m_1, s, c) = b \quad] \leq \tfrac{1}{2} + \tfrac{1}{q(\eta)}\end{aligned}$$

with the requirement that m_0 *and* m_1 *are the same length.*

(Here, the notation “$\forall$ sufficiently large η” means “$\exists \eta_0$ such that $\forall \eta \geq \eta_0$.” The notation

$$\Pr[a_1; a_2; \ldots a_n : P]$$

means the probability of predicate P being true after running experiments a_1, $a_2 \ldots a_n$ in series. Lastly, the notation $x \leftarrow D$ indicates that x is drawn from distribution D.)

Note that this definition does not specify the ability of the adversary to distinguish between the two possible encryptions (the adversary’s “advantage”) at any particular value of the security parameter, but only in the asymptotic case. In particular, it requires that the adversary’s advantage be *negligible*:

Definition 4 (Negligible) *A function* $f : \mathcal{N} \rightarrow \mathcal{R}$ *is* negligible in η *if, for any polynomial* q*,* $f(\eta) \leq \frac{1}{q(\eta)}$ *for all sufficiently large* η*. If* f *is negligible in* η*, we write* $f \leq neg(\eta)$*.*

In practice, one proves that any given algorithm meets the relevant definition of security by proving that if it did not, one could solve some underlying “hard” problem. That is, one might show

[7] Technically, the definition given here is that for a different definition of security, usually called *general message (GM) security*. However, GM security is well-known to be equivalent to semantic security, and it has a slightly more convenient form for our purposes.

that an algorithm is a secure encryption scheme by assuming that there exists an adversary $\mathtt{A}$ that is able to break it. Then one would show that there exists another PPT algorithm that uses $\mathtt{A}$ as a black-box to solve some underlying hard problem.

For example, the *decisional Diffie-Hellman* problem is one of the most widely-used intractable problems in cryptography. In this problem, a family of cyclic groups $\{G_\eta\}_\eta$ is fixed and indexed by security parameter. Consider two experiments for a given adversary $\mathtt{A}$ and value of η:

$\mathbf{Exp}_1$ The adversary is given η, G_η, a generator g, and a value drawn from the distribution

$$\mathbf{D} = \left\{g, g^a, g^b, g^{ab} : a, b \leftarrow |G|\right\}.$$

The adversary $\mathtt{A}$ returns a single bit.

$\mathbf{Exp}_2$ The adversary is given η, G_η, a generator g, and a value drawn from the distribution

$$\mathbf{R} = \left\{g, g^a, g^b, g^z : a, b, z \leftarrow |G|\right\}.$$

Again, the adversary $\mathtt{A}$ returns a single bit.

The goal of the adversary is to return a different bit in experiment $\mathbf{Exp_1}$ than in experiment $\mathbf{Exp_2}$. That is, the adversary is being asked to distinguish between the actual Diffie-Hellman value g^{ab} and a random group element g^z. The advantage of the adversary in this case is

$$\mathbf{Adv}_{\mathtt{A},\eta} = \left|\Pr\left[\mathtt{A} \text{ returns 1 in } \mathbf{Exp_1}\right] - \Pr\left[\mathtt{A} \text{ returns 1 in } \mathbf{Exp_2}\right]\right|$$

It is widely believed that for certain families of groups, the decisional Diffie-Hellman problem is hard. That is, no adversary can maintain a non-negligible advantage as the security parameter grows:

$$\forall \text{ PPT algorithms } \mathtt{A}, \mathbf{Adv}_{\mathtt{A},\eta} \leq \mathit{neg}(\eta)$$

Given this, one might implement a semantically-secure encryption scheme in the following way:

- The key generation algorithm G, on input 1^η, picks $a \leftarrow |G_\eta|$. The secret key is a and the public key is (η, g, g^a).
- The encryption algorithm E takes in the public key (η, g, g^a) and message $m \in G_\eta$. It then picks a random $b \leftarrow |G_\eta|$ and outputs the ciphertext $(g^b, m \cdot g^{ab})$.
- The decryption algorithm D, on ciphertext $(g^b, m \cdot g^{ab})$ and secret key a, first calculates $\left(g^b\right)^a = g^{ab}$, and then outputs the message $m \cdot g^{ab}/g^{ab} = m$.

This is the ElGamal encryption scheme [21], which provably reduces to the decisional Diffie-Hellman problem. That is, if an adversary `A` can break the semantic security of this scheme, then another adversary $\mathtt{A}'$ can solve the decisional Diffie-Hellman problem. The details of the proof are somewhat technical, but the basic idea is this: Adversary $\mathtt{A}'$ gets (g, g^a, g^b, h) as input and wants to know if h is the Diffie-Hellman value g^{ab} or a random group element g^z. To do this, it gives (η, g, g^a) to `A`, who interprets it as a ElGamal public key. `A` then produces two messages m_0 and m_1 and gives them to $\mathtt{A}'$. Adversary $\mathtt{A}'$ picks a random bit b randomly, and gives $m_b \cdot h$ to `A` as the ciphertext.

If the input (g, g^a, g^b, h) to $\mathtt{A}'$ was from distribution $\mathbf{D}$, then the input to `A` is exactly as expected and `A` will be able to tell that $m_b \cdot h$ is the "encryption" of message m_b. Hence, `A` will be able to guess b with non-negligible probability. If, on the other hand, the input (g, g^a, g^b, h) is from distribution $\mathbf{R}$, then the ciphertext $m_b \cdot h$ is completely independent of the messages m_0 and m_1, and `A` will not be able to guess b with probability greater than 1/2. Thus, the behavior of `A` will differ between the two types of input in a discernible way. The adversary $\mathtt{A}'$ (who knows b) can use this difference in the behavior of `A` to differentiate between input from $\mathbf{D}$ and input from $\mathbf{R}$.

As can be seen from the above proof sketch, the proofs of this model are extremely sound and specific. The above proof, for example, proves that *no* efficient algorithm can break the ElGamal encryption scheme so long as one specific computational problem remains intractable. Similar proofs exist for specific key-exchange or authentication protocols. However, even the proof-sketch above is somewhat complicated. The full proof is more complex yet, and proofs of protocols tend to be not only complex but tedious as well. Furthermore, it is not at all clear that proof techniques like the one above can be automated in any way. Presently, proofs in this model need to be 'hand-crafted', and it is difficult to prove general theorems that apply to more than one scheme or protocol.

1.5 Combining the Two

Given how these two models complement each other, it seems an enticing goal to unify them in some way to gain the benefits of both. While the computational model uses its low level of abstraction to gain strong proofs, the Dolev-Yao model leverages its high level of abstraction to provide intuitive and simple proof methods. If the assumptions of the Dolev-Yao model could be justified in some computational setting, then protocol designers could use high-level methods to yield low-level proofs.

Much work has already been done in this area, which we will review in depth in Chapter 5. Here, however, we will mention one particular result of our own [30] that will motivate the work contained in the remainder of this document. This result, building on earlier work [4], is an early attempt to show an equivalence between active computational adversaries and active Dolev-Yao adversaries. In particular, we showed (albeit in a weak way) that sufficiently strong computational cryptography can prohibit the computational adversary from producing any message that could not also be produced by the Dolev-Yao adversary.

In particular, we formalize the intuition of Assumption 3 in the language of computational cryptography, using a series of intermediate attempts. Because of the importance of this formalization, we reproduce it here:

Intuitively, we would like to say that it should be hard for the computational adversary to produce a single message outside the closure of its input. Informally:

Attempt 1 *An abstract encryption operator provides* weak Dolev-Yao non-malleability[8] *if*

$$\forall PPT \text{ adversaries } \mathtt{A}, \forall S \subseteq \mathcal{A}, \forall M \in (\mathcal{A} \setminus C[S])$$
$$\Pr[N \leftarrow \mathtt{A}(S) : N = M] \leq neg(\eta)$$

Here, $\Pr[A; B; C : P]$ indicates the probability of predicate P being true after running experiments A, B and C in series. The notation $x \leftarrow D$ indicates x being drawn from distribution D. If D is a set, the uniform distribution is used. If D is an algorithm, we use the distribution over output induced by the distribution of the input and the distribution of D's random coin flips.

Although this attempt contains the desired intuition, there are two small problems:

- It is unclear how a set S of Dolev-Yao messages can be passed as input to a computational adversary, or how a Dolev-Yao message M can be produced as output.
- It is not clear how the probability relates to the security parameter η.

The purpose of this section is to make the above definition meaningful. Our main tool for doing so will be a mapping from Dolev-Yao messages to their computational analogues: probability distributions on bit-strings. The mapping we present here is congruent to that given by Abadi and Rogaway [3, 4], adapted to the public-key encryption setting.

The "encoding" of a formal message M, written $[\![M]\!]_\eta^t$, is a probability distribution that depends on four things:

- The formal message M,
- The *tape* (t) which is an infinite sequence of bits. We assume for convenience that we have random access to this tape, although this can be easily simulated using a standard tape and some book-keeping. In usage, we will assume that the bits on this tape are random.
- The security parameter η.
- An arbitrary public-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$.

[8] *Originally called "ideal" encryption in [30]. We use the more descriptive name given here to distinguish it from the stronger notions we will consider in Section 1.8.*

Definition 5 (Encoding: messages) *Let $\eta \in \mathcal{N}$ be the security parameter. Let $t \in \{0,1\}^{\omega}$ be a random tape, partitioned into a length-η segment for each nonce and public key in $\mathcal{A}$. Let $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be a public-key encryption scheme. Then for any $M \in \mathcal{A}$, the* encoding *of M, written $[\![M]\!]_{\eta}^{t}$, is defined recursively as:*

- *If $M \in \mathcal{R}$ is a nonce, then $[\![M]\!]_{\eta}^{t} = \langle \sigma_M, \text{“nonce”} \rangle$, where σ_M is the value of the tape partition associated with M.*

- *If (M, M^{-1}) is a public/private key pair, then $[\![M]\!]_{\eta}^{t} = \langle \mathsf{e}, \text{“pubkey”} \rangle$ and $[\![M^{-1}]\!]_{\eta}^{t} = \langle \mathsf{d}, \text{“privkey”} \rangle$ where (e, d) is the output of $\mathsf{G}(1^{\eta}, \sigma_M)$. Note that we now explcitly require that the randomness of the key-generation algorithm be the value σ_M from the tape.*

- *If $M \in \mathcal{M}$ is an identifier, then $[\![M]\!]_{\eta}^{t}$ is mapped to $\langle \mu(M), \text{“id”} \rangle$ where μ is any (short) efficiently-computable functions from identifiers to bit-string. That is, we do not care how identifiers are mapped to bit-strings so long as each identifier is uniquely represented and it is efficient to compute the encoding of a given identifier.*

- *If $M = M_1 \, M_2$, then $[\![M]\!]_{\eta}^{t}$ is the mapping from pairs of distributions to distributions given by $\left\langle [\![M_1]\!]_{\eta}^{t}, [\![M_2]\!]_{\eta}^{t}, \text{“pair”} \right\rangle$.*

- *If $M = \{\![M']\!\}_K$ is an encryption, then $[\![M]\!]_{\eta}^{t}$ is the mapping from pairs of distributions to distributions given by $\left\langle \mathsf{E}\left([\![M']\!]_{\eta}^{t}, [\![K]\!]_{\eta}^{t}\right), [\![K]\!]_{\eta}^{t}, \text{“enc”} \right\rangle$*

If $S \subseteq \mathcal{A}$, then by $[\![S]\!]_{\eta}^{t}$ we mean $\left\langle [\![s_1]\!]_{\eta}^{t}, [\![s_2]\!]_{\eta}^{t}, \ldots \right\rangle$ where s_1, s_2 are the elements of S in some canonical order. By $[\![M]\!]_{\eta}$ we mean the distribution

$$\left\{ t \leftarrow \{0,1\}^{\omega} ; m \leftarrow [\![M]\!]_{\eta}^{t} : m \right\}.$$

The bits on the tape are used to represent the coin flips used to make atomic elements, and we will later enforce that the tape is filled with random bits. Compound terms are made via either bit-string concatenation or a computational encryption scheme. Note that the coin flips used by the encryption algorithm are *not* taken from the tape. Hence, $[\![\{\![M']\!\}_K]\!]_{\eta}^{t}$ remains a distribution even if t is fixed.

There are two properties of computational public-key encryption that our encoding mapping will need to accommodate and will arise later.

- First, public-key encryption is not required to hide the key used to encrypt. We make this possible leak of information explicit in the definition above by explicitly concatenating each ciphertext with the encrypting key.

- Secondly, computational public-key encryption is not generally required to hide the length of the plaintext. For this reason, we need to limit the amount of information about a plaintext that will be revealed by its length. We will assume that the length of a message depends only on the message's structure, not any of its component values. More formally, let the *type tree* of a formal message be the same as its parse tree except that each leaf is replaced by its type. We use the same notation for type trees that we do for messages. Thus, the type tree of a message $\{\!|A\,N|\!\}_K$ (where $A \in \mathcal{M}$, $N \in \mathcal{R}$ and $K \in \mathcal{K}_{Pub}$) is $\{\!|\mathcal{M}\,\mathcal{R}|\!\}_{\mathcal{K}_{Pub}}$. We assume that the length of a formal message M depends only on $\mathcal{T}_M$, the type tree of M, and the security parameter. This is not an unreasonable assumption. The above definition of the encoding mapping implies that all nonces encode to the same length. The assumption can be trivially enforced for other type trees by padding out to some maximal length. Thus, we will use $\left|[\![M]\!]_\eta^t\right|$ to designate the unique length of encodings of M.

The encoding mapping allows formal messages to be represented as bit-strings, which allows formal messages to be passed to and returned by the computational adversary. Furthermore, it uses the security parameter, showing how the probability in Attempt 1 relates to η. Thus, we can re-attempt to translate Assumption 3 into computational terms:

Attempt 2 *An encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *provides* weak Dolev-Yao non-malleability *if, when used in* $[\![\cdot]\!]_\eta^t$,

$$\begin{array}{ll} & \forall\, PPT\ \ adversaries\ \mathtt{A},\ \forall\, S \subseteq \mathcal{A},\ \forall M \in (\mathcal{A} \setminus C[S]) \\ \Pr[& t \leftarrow \{0,1\}^\omega\ ; \\ & s \leftarrow [\![S \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Adv} \cup \mathcal{R}_{Adv} \cup \mathcal{M}]\!]_\eta^t\ ; \\ & m \leftarrow \mathtt{A}(1^\eta, s)\ : \\ & m \in supp\ [\![M]\!]_\eta^t \qquad\qquad\qquad] \leq neg(\eta) \end{array}$$

Here, *supp* D means the support of distribution D. When the support of a distribution contains one element, we will treat the distribution itself as a singleton set.

This definition is still problematic, however, for two technical reasons. First, the input to the adversary might be of infinite length. The set S may be of infinite length, or there may be an infinite number of elements in $\mathcal{M}$, $\mathcal{R}_{Adv}$, $\mathcal{K}_{Pub}$ and $\mathcal{K}_{Adv}$. If any of these are the case, then the restriction of the adversary to probabilistic polynomial-time is meaningless. No computational encryption scheme would remain secure against an infinite-time adversary. For this reason, we require that S be of finite size. The sets $\mathcal{M}$, $\mathcal{R}_{Adv}$, $\mathcal{K}_{Pub}$ and $\mathcal{K}_{Adv}$ might still be infinite, so instead of passing them as input we represent them via oracles:

- $\mathtt{M}_\eta^t(x)$ returns (the encoding of) the identifier of the xth participant.
- $\mathtt{R}_\eta^t(x)$ returns the (encoding of) the xth nonce in $\mathcal{R}_{Adv}$,
- $\mathtt{PbK}_\eta^t(x)$ returns the public key of principal x, and
- $\mathtt{PrK}_\eta^t(x)$ returns the private key of x of $x \in [\![K^{-1}]\!]_\eta^t$ if $K^{-1} \in \mathcal{K}_{Adv}$.

The second problem is that our results rely upon a technical limitation: acyclicity of encryptions. A set of encryptions is acyclic if, when K_1 encrypts K_2^{-1} in some element of S, and K_2 encrypts K_3^{-1}, and so on, this sequence of keys encrypting keys never loops back on itself. More formally:

Definition 6 (Acyclic) *For an expression M, construct a graph G_M where the nodes are the public/private key pairs used in the expression. We draw an edge from $p_1 \to p_2$ if in M the private key K_2^{-1} associated with pair p_2 is encrypted with K_1, the public key associated with p_1. The expression M is* acyclic *if the graph G_M is acyclic.*

For example, the message

$$\{\!|K_1^{-1}|\!\}_{K_2}\ \{\!|K_2^{-1}|\!\}_{K_3}$$

is acyclic, but the message

$$\{\!|K_1^{-1}|\!\}_{K_2}\ \{\!|K_2^{-1}|\!\}_{K_1}$$

is not. Our results will only hold for acyclic sets S. However, protocols analyzed in the Dolev-Yao model typically operate in one of three ways:

- Long-term keys are used to encrypt session keys, which themselves never encrypt other keys,
- The present session key is used to encrypt the next session key, but never the previous, or
- Keys are never encrypted at all.

None of these cases will produce cyclic encryptions.

Thus, we arrive at our final security condition:

Definition 7 (Dolev-Yao weak non-malleability) *An encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *provides* weak Dolev-Yao non-malleability *if, when used in* $[\![\cdot]\!]_\eta^t$,

$$\begin{aligned}
&\forall\, PPT \text{ adversaries } \mathtt{A},\ \forall \text{ acyclic finite } S \subseteq \mathcal{A},\ \forall M \notin C[S]\,,\\
&\Pr[\quad t \leftarrow \{0,1\}^\omega\\
&\qquad s \leftarrow [\![S]\!]_\eta^t\,;\\
&\qquad m \leftarrow \mathtt{A}^{M_\eta^t(\cdot), PbK_\eta^t(\cdot), PrK_\eta^t(\cdot), R_\eta^t(\cdot)}(1^\eta, s)\,:\\
&\qquad m \in supp\ [\![M]\!]_\eta^t \qquad\qquad\qquad\qquad] \leq neg(\eta)
\end{aligned}$$

The main result of [30] is that this definition can be satisfied by a standard definition of security in the computational world: *plaintext-aware* cryptography.

1.6 Plaintext-Aware Encryption

Intuitively, a public-key encryption scheme is plaintext-aware [10, 7] if it reveals nothing about the plaintext to encryptions created by honest principals, but ensures that the adversary knows the

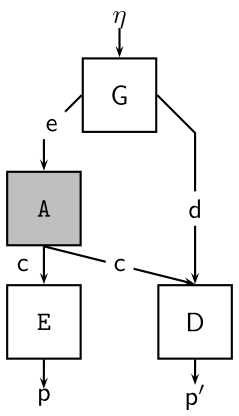


Figure 1-2: The basic plaintext-awareness ”game” (attempt)

plaintext to any ciphertext it creates. This intuition is represented by insisting that there exist an "extractor" E that can produce the plaintext of all ciphertexts of adversarial origin. That is, the extractor is going to play the following game (also shown in Figure 1-2):

- A key pair (e, d) is chosen honestly and randomly according to the key-generation algorithm G.
- The adversary A is given e and allowed to perform any efficient computation. At the end, it must produce a ciphertext c.
- The extractor E is given e and c, and must produce a candidate plaintext g.

The extractor "wins" if the guess g is also what would have been produced by decrypting c with d. An encryption scheme is plaintext aware if there exists an extractor that can win against every adversary (almost) all the time. If such an adversary exists, then the adversary must know (or rather, be able to know by running the extractor) the plaintext to every ciphertext it creates.

To formalize (incorrectly) this intuition:

Attempt 3 *A public-key encryption algorithm* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is* plaintext-aware *if it is semantically secure and:*

$$\begin{aligned} &\exists\, \mathtt{E}_{PPT}, \forall\ PPT\ algorithms\ \mathtt{A}, \\ &\Pr[\quad (\mathsf{e},\mathsf{d}) \leftarrow \mathsf{G}(1^\eta); \\ &\qquad \mathsf{c} \leftarrow \mathtt{A}(1^\eta, \mathsf{e}); \\ &\qquad g \leftarrow \mathtt{E}(1^\eta, \mathsf{d}, c) : \\ &\qquad g = \mathsf{D}(c, \mathsf{d}) \qquad] \geq 1 - neg(\eta) \end{aligned}$$

However, the above definition is a contradiction. There cannot be such an extractor for a semantically-secure encryption scheme. If there were, then the adversary in Definition 3 could simply use it to decrypt and learn whether the challenge ciphertext c contained message m_0 or m_1 as a plaintext.

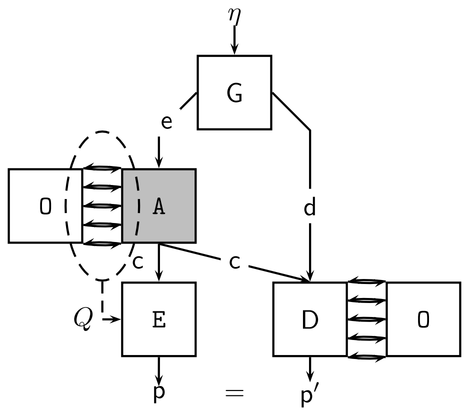


Figure 1-3: The plaintext-awareness game (final)

However, all is not lost. The resolution of this difficulty is to limit somehow the extractor to ciphertext of the adversary's creation. To do this, we define the extractor to use some extra information that the adversary can provide about its own ciphertexts, but cannot also provide about the ciphertexts of honest participants.

To accomplish this, the encryption algorithm is redefined to use a *random oracle* $\mathtt{O}(\cdot)$: an oracle that provides a random mapping from inputs to bit-strings of sufficient length. The adversary is also given access to this oracle, but the extractor is now given (as the additional information) all the oracle queries made by the adversary (as shown in Figure 1-3:

Definition 8 (Plaintext-Aware Encryption) *An encryption scheme* $(\mathsf{G}^{\mathsf{O}}, \mathsf{E}^{\mathsf{O}}, \mathsf{D}^{\mathsf{O}})$ *is* plaintext-aware *if it is semantically secure and*

$$
\begin{aligned}
&\exists\, \mathtt{E}_{PPT}, \forall \textit{ PPT algorithms } \mathtt{A}, \\
&\Pr[\quad (\mathsf{e}, \mathsf{d}) \leftarrow \mathsf{G}(1^\eta); \\
&\qquad c, Q \leftarrow \mathtt{A}^{\mathtt{O}(\cdot)}(1^\eta, \mathsf{e}); \\
&\qquad g \leftarrow \mathtt{E}(1^\eta, \mathsf{e}, c, Q) : \\
&\qquad g = \mathsf{D}^{\mathsf{O}(\cdot)}(c, \mathsf{d}) \qquad] \geq 1 - neg(\eta)
\end{aligned}
$$

where Q is a transcript of every query and response communicated between $\mathtt{A}$ *to* $\mathtt{O}(\cdot)$*.*

Giving the extractor access to the adversary's oracle queries in the above definition makes it significantly more powerful than the adversary in Definition 3, which presumably does not see the oracle queries made by honest parties. Thus, the adversary can simply run the extractor to learn the plaintexts of its own ciphertexts, but cannot run the extractor to decrypt the ciphertexts of honest participants.

1.7 Our Previous Results

Plaintext-aware encryption, it turns out, is sufficiently strong to satisfy Definition 7. As we showed in [30]:

Theorem 1 *A plaintext-aware encryption scheme provides weak Dolev-Yao non-malleability.*

Thus, sufficiently strong computational cryptography can enforce a connection between the formal and computational worlds. However, Theorem 1—while being true—has a number of weaknesses:

- The formalization of the Dolev-Yao adversary in Definition 7 is weak. Informally, it states that the adversary cannot create a given message M. That is, the ability of the adversary to hit *one* particular target is small. However, the Dolev-Yao algebra is infinite; one would instead wish to know that the adversary has only negligible probability of hitting *any* valid target.

- Plaintext-aware encryption requires the use of a random oracle, and hence has been considered somewhat suspect. The random oracle model was originally proposed as an abstraction for such algorithms as SHA-1 [54] or HMAC [35]. The idea was to use the random oracle when proving properties of higher-level schemes, but to replace the external oracle with internal calls to these algorithms when creating the implementation. However, this does not seem feasible for two reasons:

    1. Recent work [16, 29] indicates that this substitution may not be even possible in general.
    2. The random oracle is used in the definition of plaintext awareness to give the extractor a "window" into the internal state of the adversary (as revealed through its queries). If the external random oracle is replaced by an internal algorithm, then this window is closed.

    Thus, it seems that plaintext awareness requires the use of an external oracle.

    However, any plaintext-aware encryption scheme that uses communication with an actual oracle must be both inefficient and untrustworthy. It must be inefficient because both encryption and decryption require communication with the oracle, incurring an unacceptable communication overhead. It must also be untrustworthy because the scheme is secure only if the adversary does not learn the oracle queries. Thus, a plaintext-aware encryption scheme that uses an oracle is only secure if the oracle is honest and communication with it is confidential—two strong and untenable assumptions.

    Hence, the use of the random oracle has prevented the notion of plaintext awareness from receiving widespread acceptance as a viable goal.

Thus, Theorem 1 shows only that a weak version of the Dolev-Yao assumptions can be satisfied by a problematic definition of computational security.

1.8 Results of This Thesis

In this work, we strengthen Theorem 1 in four ways.

- Firstly, we investigate stronger versions of the Dolev-Yao assumptions, Assumption 3 in particular. While Definition 7 fixes the target Dolev-Yao message M, one could consider stronger definitions that do not. For example, one might consider a definition which stipulates that the adversary cannot create a message outside the encoding, even if the adversary itself gets to choose the message to be created. Stronger yet, one might want to show that no matter what the adversary outputs, it will not be the encoding of any message outside the closure. We call this intuition *strong* Dolev-Yao non-malleability:

  Attempt 4 *An abstract encryption operator provides* strong *Dolev-Yao non-malleability if*

$$\begin{array}{ll} \multicolumn{2}{l}{\forall\, \mathtt{A}_{PPT},\ \forall\, \textit{finite, acyclic } S \subseteq \mathcal{A} :} \\ \Pr[& t \leftarrow \{0,1\}^{\omega} \\ & s \leftarrow [\![S]\!]_{\eta}^{t}\,; \\ & m \leftarrow \mathtt{A}^{\mathcal{M}_{\eta}^{t}(\cdot),\mathcal{PbK}_{\eta}^{t}(\cdot),\mathcal{PrK}_{\eta}^{t}(\cdot),\mathcal{R}_{\eta}^{t}(\cdot)}(1^{\eta}, s) : \\ & \exists M \in \mathcal{A} \setminus C[S]\, .m \in \textit{supp}\ [\![M]\!]_{\eta}^{t} \qquad\qquad] \leq \textit{neg}(\eta) \end{array}$$

  (This differs from Definition 7 in that instead of being universally quantified outside the probability, M is existentially quantified in the predicate defining success for the adversary.)

  Unfortunately, the above property may not be realizable. As we note above, the Dolev-Yao model is not necessarily restricted to finite algebras. (It is for this reason that the adversary's knowledge of e.g., $\mathcal{R}_{Adv}$ was represented via an oracle in Definition 7.) If the Dolev-Yao algebra contains an infinite number of nonces, for example, then every bit-string of the appropriate length will be the encoding of an infinite number of nonces. Hence, the adversary could be virtually ensured of creating the encoding of *some* honest party's nonce by outputting a random bit-string of nonce length.

  The central core of this difficulty is that Assumption 1 (unique representation of messages) cannot hold even approximately when the Dolev-Yao algebra is infinite but its computational encoding must fit in a finite space. We consider two possible and mutually consistent resolutions. The first and simpler resolution is to require the Dolev-Yao algebra to be finite. The second possibility is to somehow give each Dolev-Yao message a unique representation in the computational setting. We have explored these possibilities in depth and have produced from them two realizable definitions of strong Dolev-Yao non-malleability (Definitions 10 and 12 of Section 2).

- Theorem 1 shows that one notion of security (weak Dolev-Yao non-malleability) is satisfied by another (plaintext awareness of encryption). There are two ways in which such a statement can be strengthened. First, one can strengthen the satisfied notion, which we discuss above. The second is to weaken the satisfying notion, and it is imperative that this be done for Theorem 1. As we have previously mentioned, plaintext awareness is rightfully considered suspect due to its reliance on the random oracle. So long as Dolev-Yao non-malleability relies on plaintext awareness, and hence the random oracle, it will be considered suspect as well.

  One of two things must be done. Either Dolev-Yao non-malleability must be shown to be satisfied by other definitions of encryption security, or plaintext awareness must be re-defined in a way that does not use the random oracle. In this work, we do both.

- First, we show that Dolev-Yao non-malleability can be satisfied by a weaker, more generally-accepted notion of encryption security called *security against the chosen ciphertext attack* (Definition 13). This notion has been widely studied [57, 52, 59] and there exist efficient implementations [17]. We show that strong Dolev-Yao non-malleability—both the finite-algebra version (Definition 10) and the infinite-algebra version (Definition 12)—are satisfied by chosen-ciphertext security (Theorem 5).

- We also propose another definition of plaintext awareness that does not rely on the random oracle (Section 3.1.2). This definition replies upon an alternate model of public-key encryption in which both the sender and the receiver have public keys. Furthermore, both the sender and the receiver must register their public keys with a trusted third party called the *registration authority*. (This is not an unreasonable model. In practice, users already register their keys with a certification authority.)

  Plaintext awareness in this setting requires that the adversary must know (meaning that an extractor can extract) the plaintext of any ciphertext it creates, so long as the nominal sender's keys have been registered. That is, if a party P has registered their public keys with the registration authority, then whenever the adversary creates a ciphertext ostensibly from P it will also know the plaintext. However, we require that our scheme tolerate the corruption of the trusted third party in that it remain chosen-ciphertext secure even when the registration authority is subverted by the adversary.

  We explain this new model and definition at length in Section 3.1. We also provide an implementation for this new definition (Section 3.3) which is conceptually simple and relies only on general assumptions. That is, it only assumes other standard cryptographic definitions and does not rely upon particular intractability assumptions. In fact, this scheme can be thought of as a slight variant to the first known implementation of chosen-ciphertext security [59].

- Lastly, we begin to look beyond the standard Dolev-Yao model. As is typically presented, the Dolev-Yao model contains only one operator which is cryptographic in nature: encryption. There exist variants in the literature that contain operators for signatures and hashing, as well. However, the most popular protocols in real-world practice are not captured by these, and will not until the Dolev-Yao model is expanded to include the Diffie-Hellman key-agreement scheme.

  However, this is more easily said than done. As opposed to encryption and signing, for example, the Diffie-Hellman scheme is not an abstract definition but a specific number-theoretic algorithm. The security of this scheme depends, in turn, upon a specific computational assumption. The issue here is not whether an established formal model can be made computationally sound, but how best to reflect a computational assumption in a formal setting. The Diffie-Hellman scheme relies upon a non-free operation, as opposed to the traditionally free Dolev-Yao algebra. Also, it is not immediately clear what the powers of the formal adversary should be—any efficient computation is possible.

  We consider these issues at length in Chapter 4. In particular, we formulate a reasonable way to express the Diffie-Hellman scheme in the Dolev-Yao framework (Definition 34). We also consider the issues of non-freeness that are inevitably raised, and show how to represent the opportunities they present to the adversary (Definition 37).

  We then show how the Diffie-Hellman assumption (which provides security to the similarly-named key-agreement scheme) can be represented in the Dolev-Yao model. This property (Condition $\mathcal{DH}$, Definition 40) is strong enough to enable proofs in the Dolev-Yao model. We also show that if there exists a Dolev-Yao execution that violates it, there exists an adversary that can violate the (computational) Diffie-Hellman assumption.

  However, this is not enough to guarantee computational soundness. The adversary so produced may not be efficient in its execution, and thus would not violate the Diffie-Hellman assumption. This possibility results from the fact that protocols in the Dolev-Yao model do not *a priori* need to be efficiently executable. Thus, the execution of the Dolev-Yao protocol may not be efficiently executable either.

  We resolve this by considering only the sub-class of "silent" Dolev-Yao protocols (Definition 41). In these protocols, honest participants do not use the secret Diffie-Hellman values as plaintexts, but only as keys. We show that these protocols must be efficiently executable if the underlying cryptographic algorithms are sufficiently strong. In fact, we use the same strong computational assumption that enabled Theorem 1: the random oracle. If the random oracle exists, we show that condition $\mathcal{DH}$ is computationally sound for silent Dolev-Yao protocols. (Just as the random oracle will be removed from Theorem 1 in Chapters 2 and 3.1, we will consider in Section 5.3 possible ways by which it could be removed from this result also.)

Chapter 2

Stronger Dolev-Yao Properties

In this section, we strengthen the result of [30] in two important ways:

- We strengthen the computational interpretation of the Dolev-Yao model beyond that contained in Definition 7. In particular, we propose two new notions of Dolev-Yao non-malleability. Whereas weak Dolev-Yao non-malleability considered the probability that the adversary can create a given, fixed message, our new notions consider the probability that the adversary can create *any* message. The first of these notions (*finite Dolev-Yao non-malleability*)considers the case where the algebra has a finite number of elements, as one would encounter in the setting or a bounded number of participants engaging in a bounded number of protocol executions. The second notion (*infinite Dolev-Yao non-malleability* considers the case of an infinite algebra, and requires the adversary to not only produce a message but also to know *which* message is created.

- We show that our new definitions can be satisfied by computational encryption strictly weaker than plaintext-awareness. In particular, we show that both notions are satisfied by encryption secure against the *chosen-ciphertext attack* (also known as *chosen-ciphertext security*). This definition of security is like that of semantic security, except that the adversary has (slightly limited) access to a decryption oracle. It does not use a trusted third party, and hence is not susceptible to the same criticisms as plaintext-awareness.

  The proof that chosen-ciphertext security satisfies strong Dolev-Yao non-malleability will first use an indistinguishability lemma like that of Abadi and Rogaway[4], which is mildly interesting in its own right.

2.1 New Definitions

As mentioned above, the security property in Definition 7 concerns only the probability that the adversary can produce some given "bad" message M. However, what does this tell us about the ability of the adversary to produce *some* bad message of its choice? The simplest approach to this question is to use Definition 7 and a union bound. Intuitively (and in the style of Assumption 1) the union bound tells us that:

$$\begin{aligned}
\Pr\left[\mathtt{A}(S) \text{ produces an } M \in \mathcal{A} \setminus C[S]\right] &\geq \sum_{M \in \mathcal{A} \setminus C[S]} \Pr\left[\mathtt{A}(S) \text{ produces } M\right] \\
&\geq \sum_{M \in \mathcal{A} \setminus C[S]} \epsilon \\
&= \epsilon \left|\mathcal{A} \setminus C[S]\right|
\end{aligned}$$

Even though Definition 7 tells us that ϵ is negligible, this is not enough to show that $\epsilon \left|\mathcal{A} \setminus C[S]\right|$ is also negligible. In particular, there are as yet no assumptions that limit the size of $|\mathcal{A} \setminus C[S]|$. Consider, for example, the nonces. There is no requirement yet that the set of nonces $\mathcal{R}$ be finite, much less a function of the security parameter. Thus, $\mathcal{R} \setminus \mathcal{R}_{Adv}$ might be infinite also, as well as the number of nonces in $\mathcal{A} \setminus C[S]$.

One can see the difficulty in another way. Recall that the encoding of a nonce is chosen uniformly from bit-strings of length η. If there are an infinite number of nonces, then *any* bit-string of length η is guaranteed to be the encoding of an infinite number of them, including an infinite number in $\mathcal{R} \setminus \mathcal{R}_{Adv}$ (if the tape t is chosen randomly). Thus, by choosing a random string of the right length, the adversary is guaranteed to produce the encoding of some "bad" nonce.

There are a number of ways around this difficulty. First, we can simply restrict the set of nonces, and other aspects of the algebra, to polynomial size. If the number of nonces is polynomial in the security parameter, then a random bit-string will have only a negligible probability of being the valid encoding of a nonce. To formalize:

Definition 9 *Let $\mathcal{A}_\eta$ be the smallest free algebra where atomic terms are of four types:*

1. *Names ($\mathcal{M}$)*
2. *Nonces ($\mathcal{R}_\eta$)*
3. *Public keys ($\mathcal{K}_{Pub\eta}$), and*
4. *Private keys ($\mathcal{K}_{Priv\eta}$)*

where $\mathcal{R}_\eta$, $\mathcal{K}_{Pub\eta}$, $\mathcal{K}_{Priv\eta}$ are all of size η, and compound terms are created via

- *$encrypt : \mathcal{K}_{Pub} \times \mathcal{A} \rightarrow \mathcal{A}$*

- *pair* : $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$

Intuitively, restricting ourselves to a finite algebra such as this is not too onerous a restriction: it might simply reflect a bounded setting with a polynomial number of participants willing to engage in the protocols a polynomial number of times.[1] One might also expect that the number of names be bounded by η for the same reason, or the size of compound messages. These other restrictions might also follow from the bounded setting, but it turns out that we will not require them in our proofs.

If we restrict ourselves to this finite algebra, we can strengthen our notion of Dolev-Yao security in the natural way:

Definition 10 *A computational encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *provides* strong finite Dolev-Yao non-malleability *if, when used in* $[\![\cdot]\!]_\eta^t$*:*

$$\begin{array}{ll} \forall\, \mathtt{A}_{PPT}, \forall \textit{ finite, acyclic } S \subseteq \mathcal{A}_\eta : & \\ \Pr[\quad t \leftarrow \{0,1\}^\omega & \\ \quad s \leftarrow [\![S]\!]_\eta^t\,; & \\ \quad m \leftarrow \mathtt{A}^{M_\eta^t(\cdot), PbK_\eta^t(\cdot), PrK_\eta^t(\cdot), R_\eta^t(\cdot)}(1^\eta, s) : & \\ \quad \exists M \in \mathcal{A}_\eta \setminus C[S]\,.m \in \mathit{supp}\ [\![M]\!]_\eta^t &] \leq \mathit{neg}(\eta) \end{array}$$

Another way to side-step the problem is to allow the algebra to remain infinite, but to require more from the adversary. We would like to ensure that a random guess is unlikely to be a valid message. However, this seems difficult. So long as the encoding algorithm maps an infinite number of formal messages to a finite space of bit-strings, any element of that space will be the valid encoding of *some* message. However, it seems reasonable to require the adversary to not only create a message, but to also know which message it created.

To formalize this, we introduce the notion of a *tag*, which will serve to identify a message without revealing the values of its components:

Definition 11 *Let* $t(\cdot) : \mathcal{A} \to \{0,1\}^*$ *be the function where:*

- *For* $N \in \mathcal{M}$*,* $t(N) = \langle \text{"name"}, \mu(N) \rangle$*, where* μ *is the function used to encode names in the encoding mapping,*

- *For* $N \in \mathcal{R}$*,* $t(N) = \langle \text{"nonce"}, l_n(N) \rangle$ *where* $l_n(N)$ *produces a finite machine-readable label uniquely identifying the nonce* N*, where* $l(N)$ *is completely independent of the distribution* $[\![N]\!]_\eta$*.*

- *For* $K \in \mathcal{K}_{Pub}$*,* $t(N) = \langle \text{"pubkey"}, l_k(N) \rangle$ *where* $l_k(N)$ *produces a finite machine-readable label uniquely identifying the key* K*, where* $l_k(K)$ *is completely independent of the distribution* $[\![K]\!]_\eta$*.*

[1] Purely computational approaches to protocol analysis often make exactly this assumption. See [9] for an example.

- *For $K \in \mathcal{K}_{Priv}$, $t(N) = \langle$ "privkey"$, l_k(N)\rangle$ where $l_k(N)$ produces a finite machine-readable label uniquely identifying the key K, where $l_k(K)$ is completely independent of the distribution $[\![K]\!]_\eta$.*
- *For the pair term $M\,N$, $t(M\,N) = \langle$ "pair"$, t(M), t(N)\rangle$*
- *For the encryption term $\{\!|M|\!\}_K$, $t(\{\!|M|\!\}_K) = \langle$ "enc"$, t(M), t(K)\rangle$*

For example, the tag of the message $\{\!|A\,N_a|\!\}_{K_B}$ (the first message in the Needham-Schroeder protocol) would be:

$$t\left(\{\!|A\,N_a|\!\}_{K_B}\right) = \Big\langle \text{“enc”}, \big\langle \text{“pair”} \left\langle \text{“name”}, \mu(A)\right\rangle, \left\langle \text{“nonce”}, l_n(N_a)\right\rangle \big\rangle, \left\langle \text{“key”}, l_k(K_B)\right\rangle \Big\rangle$$

The tag of a message provides the message's entire parse tree, except that the leaves are replaced with identifiers that do not reveal the true value of non-name leaves. Therefore, a tag can serve as an identifier for a message, and the tags of two different messages will show how they relate.

To use this in our Dolev-Yao security condition, we give the adversary a set S of messages and their tags. We require that the adversary not only produce a "bad" message M, but indicate that it knows which message it produced by also producing M's tag:

Definition 12 *A computational encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *provides* strong infinite Dolev-Yao non-malleability *if, when used in* $[\![\cdot]\!]^t_\eta$*:*

$$\begin{aligned}
&\forall\, \mathtt{A}_{PPT},\ \forall \textit{finite, acyclic } S \subseteq \mathcal{A} :\\
&\Pr[\quad t \leftarrow \{0,1\}^\omega\\
&\qquad s \leftarrow [\![S]\!]^t_\eta\,;\\
&\qquad m, \tau \leftarrow \mathtt{A}^{M^t_\eta(\cdot), PbK^t_\eta(\cdot), PrK^t_\eta(\cdot), R^t_\eta(\cdot)}(1^\eta, s, t(S)) :\\
&\qquad \exists M \in \mathcal{A} \setminus C[S]\,.\tau = t(M) \ \textit{ and } m \in \textit{supp } [\![M]\!]^t_\eta \quad] \le \textit{neg}(\eta)
\end{aligned}$$

Here, the oracles PbK^t_η, PrK^t_η and R^t_η return both a value and the appropriate tag.

As in Theorem 1, we will show that both of these security conditions—finite and infinite strong Dolev-Yao non-malleability—can be satisfied by sufficiently strong computational cryptography (Theorem 5). However, we here address the second strengthening of Theorem 1. Instead of using plaintext-aware encryption, we will use the more commonly accepted definition of security against the chosen-ciphertext attack:

A computational public-key encryption scheme provides *security against the chosen-ciphertext attack*[2] (also written *CCA-2 security* in the notation of [7]) if no adversary has a chance significantly

[2]See [59], which builds on the work of [52]. See also [17] for a practical implementation. Technically, we give the definition for *indistinguishability* under the chosen-ciphertext attack. However, since this is provably equivalent to all other notions of security under the chosen-ciphertext attack, we will simply use the more generic term of "security."

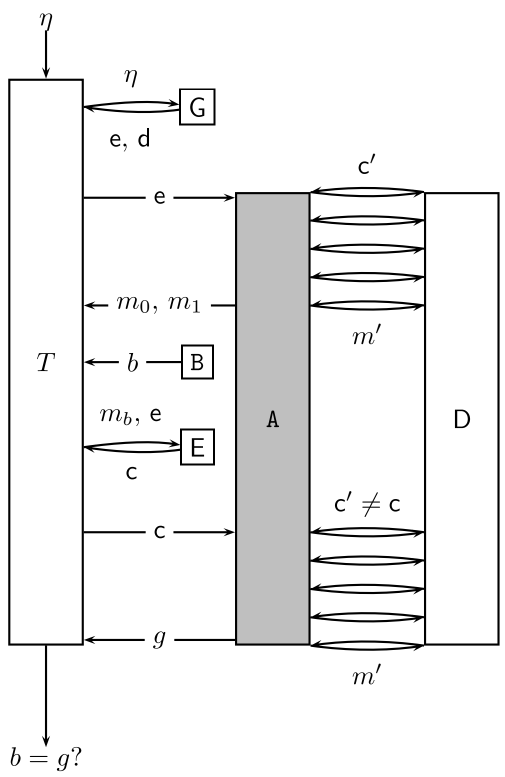


Figure 2-1: The chosen-ciphertext security game

better than random of determining accurately whether a ciphertext c is the encryption of message m_0 or message m_1, even if:

- the adversary chooses m_0 and m_1 itself, after seeing the given public key, and
- the adversary can access a decryption oracle both before choosing the messages and after receiving the ciphertext in question. (The decryption oracle will not decrypt c itself, however.)

That is, the chosen-ciphertext security "game" is exactly like that of semantic security, except that the adversary now also has almost-unlimited access to a decryption oracle (as shown in Figure 2-1). Chosen-ciphertext security means that no adversary can win this game with probability better than a random guess:

Definition 13 (Chosen-ciphertext security) *A computational public-key encryption scheme*

$(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *provides* indistinguishability under the chosen-ciphertext attack *if*

$$\begin{array}{lll} & \forall\, PPT\ \textit{adversaries}\ \mathtt{A}: & \\ \Pr[& (\mathsf{e}, \mathsf{d}) \leftarrow \mathsf{G}(1^\eta); & \\ & m_0, m_1 \leftarrow \mathtt{A}^{\mathsf{D}_1(\cdot)}(\mathsf{e}); & \\ & i \leftarrow \{0,1\}; & \\ & \mathsf{c} \leftarrow \mathsf{E}(m_i, \mathsf{e}); & \\ & g \leftarrow \mathtt{A}^{\mathsf{D}_2(\cdot)}(c): & \\ & b = g &] \leq \frac{1}{2} + neg(\eta) \end{array}$$

The oracle $\mathsf{D}_1(x)$ *returns* $\mathsf{D}(x, \mathsf{d})$*, and* $\mathsf{D}_2(x)$ *returns* $\mathsf{D}(x, \mathsf{d})$ *if* $x \neq c$ *and returns* $\perp$ *otherwise. The adversary is assumed to keep state between the two invocations. It is required that* m_0 *and* m_1 *be of the same length.*

In this chapter, we will show that both finite and infinite strong Dolev-Yao non-malleability can be satisfied by computational encryption satisfying chosen-ciphertext security. First, however, we will prove a useful indistinguishability lemma.

2.2 An Indistinguishability Lemma

In this section, we consider the indistinguishability-based definitions of Dolev-Yao security originally derived by Abadi and Rogaway [3, 4]. Intuitively, the definition of that paper describes when two formal messages should "look" the same to the formal adversary. A formal adversary has the power to make certain, limited deductions from formal messages; two given formal messages should "look" the same when all possible deductions that can be made about them yield the same results. In particular, the formal adversary of [3, 4] is assumed to be unable to distinguish between two different encryptions (unless it has the corresponding private key or keys). For example, if the adversary of [3, 4] has no other information, the two messages

$$\{\!|\{\!|A|\!\}_{K_2}\ B|\!\}_{K_1}\ K_1^{-1} \quad \text{and} \quad \{\!|\{\!|C\, D|\!\}_{K_3}\ B|\!\}_{K_1}\ K_1^{-1}$$

should be indistinguishable to it no matter what A, B, C and D are.

The fundamental result of Abadi and Rogaway is that if the encoding algorithm uses sufficiently strong computational encryption, then two messages indistinguishable to the formal adversary will encode to distributions indistinguishable to the computational adversary. Their result applies to the case of symmetric encryption, and we will here translate it to the case of public-key encryption. This translation will simultaneously strengthen and weaken the result. Indistinguishability in the public-key setting requires a stronger similarity between messages than was necessary in the case of symmetric encryption. However, our results will be able to tolerate the presence of a previously-absent strong decryption oracle.

Let T be a set of keys and suppose that the formal adversary can decrypt with regard to them. Then we represent the information that such an adversary can deduce from a formal message by its *public-key pattern*[3]:

Definition 14 (Public-key pattern) *Let $T \subseteq \mathcal{K}_{Pub}$. We recursively define the function $p(M, T)$ to be:*

- $p(K, T) = K$ *if* $K \in \mathcal{K}$
- $p(A, T) = A$ *if* $A \in \mathcal{M}$
- $p(N, T) = N$ *if* $N \in \mathcal{R}$
- $p(N_1\, N_2, T) = p(N_1, T)\, p(N_2, T)$
- $p(\{\!|M|\!\}_K, T) = \begin{cases} \{\!|p(M,T)|\!\}_K & \textit{if } K \in T \\ \langle\!|\mathcal{T}_M|\!\rangle_K & \textit{o.w. (where } \mathcal{T}_M \textit{ is the type tree of } M\textit{)} \end{cases}$

Then $\mathit{pattern}_{pk}(M, T)$*, the* public-key pattern *of an expression M relative to the set T, is*

$$p(M, \mathcal{K}_{Pub} \cap C[\{M\} \cup T]).$$

If $S \subseteq \mathcal{A}$ is a set of messages, then $\mathit{pattern}_{pk}(S, T)$ *is*

$$\langle p(s_1, C[S \cup T]), p(s_2, C[S \cup T]), \ldots \rangle$$

where s_1, s_2, ... are the elements of S is some canonical order. The base pattern *of a message M, denoted* $\mathit{pattern}_{pk}(M)$*, is defined to be* $\mathit{pattern}_{pk}(M, \emptyset)$*, and* $\mathit{pattern}_{pk}(S)$ *is defined to be* $\mathit{pattern}_{pk}(S, \emptyset)$*.*

The grammar/algebra for patterns is exactly that of messages, with the addition of a new kind of leaf node: $\langle\!|\mathcal{T}_M|\!\rangle_K$ (a "blob" of type-tree $\mathcal{T}_M$ under key K) which represents undecipherable encryptions. Unlike the "blobs" of the symmetric-encryption patterns of [3, 4], these "blobs" are labeled with K and $\mathcal{T}_M$. This is because computational encryption schemes do not necessarily hide either the encrypting key or the plaintext length.

For convenience, we define a useful relationship between two patterns:

Definition 15 (Ingredient) *If M, M' are two patterns, then M is an* ingredient *of M', written $M \sqsubseteq M'$, if the parse tree of M is a sub-tree of the parse tree of M'.*

[3]We will use "pattern" to indicate public-key pattern, as opposed to the stronger, symmetric-key definition of "pattern" in [4].

We note that since messages are special forms of patterns, this relationship can be applied between two messages as well as between a message and a pattern. We also note a relationship between a message and its pattern:

Theorem 2 *If M, M' are messages and $M' \sqsubseteq \mathit{pattern}_{pk}(M)$, then $M' \in C[M]$.*

Proof. Suppose that $M' \sqsubseteq \mathit{pattern}_{pk}(M)$. Consider the same path from root to M' in the parse tree of M. Along this path, if an interior node (not itself M') is in $C[M]$ then both child nodes are in $C[M]$:

- $C[M]$ is closed under separation. Hence, if a node is the pair $N\,N'$ and the node is in $C[M]$, then both N and N' are in $C[M]$.
- The two children of a node $\{\!|N|\!\}_K$ are N and K. Since $K \in \mathcal{K}_{Pub}$, $K \in C[M]$ automatically. Furthermore, $K^{-1} \in C[M]$ as well: if it were not, then this node of M's parse tree would have been replaced with $\langle\!|\mathcal{T}_N|\!\rangle_K$ in the parse tree of $\mathit{pattern}_{pk}(M)$. But $\langle\!|\mathcal{T}_N|\!\rangle_K$ is not a message and will not contain M' in its parse tree. So $K^{-1} \in C[M]$, and since $C[M]$ is closed under decryption with keys it contains, $N \in C[M]$.

Since the root of this path, M itself, is in $C[M]$ by definition, it must be the case that every child of every node above M' in the parse tree of M is in the set $C[M]$. Hence, $M' \in C[M]$ as well. ■

We can extend the encoding operation to the pattern algebra:

Definition 16 (Encoding: patterns) *Let:*

- *$[\![\langle\!|M|\!\rangle]\!]^t_\eta$ be any fixed bit-string of length $\left|[\![M]\!]^t_\eta\right|$ such as the all-zero string, and*
- *$[\![\langle\!|M|\!\rangle_K]\!]^t_\eta$ be the the mapping from distributions to distributions given by*

$$\left\langle \mathsf{E}\left([\![\langle\!|M|\!\rangle]\!]^t_\eta, [\![K]\!]^t_\eta\right), [\![K]\!]^t_\eta, \text{“}enc\text{”}\right\rangle.$$

Patterns allow us to state when two messages appear to be the same to the formal adversary: when they have the same pattern. The standard definition of ‘appears to be the same’ in the world of computational encryption is that of *computational indistinguishability.* We present a more general definition, which incorporates the possibility of an oracle:

Definition 17 (Computational indistinguishability) *Suppose that $\{D_\eta\}_\eta$ and $\{D'_\eta\}_\eta$ are two families of distributions indexed by the security parameter. Then they are* computationally indistinguishable with respect to a family of oracles $\mathtt{O}_x$, *written $D_\eta \cong_{\mathtt{O}_x} D'_\eta$, if*

$$\begin{array}{l} \forall\, PPT \text{ adversaries } \mathtt{A}: \\ \quad \left|\Pr[d \leftarrow D_\eta : 1 \leftarrow \mathtt{A}^{\mathtt{O}_d(\cdot)}(d, 1^\eta)] - \Pr[d \leftarrow D'_\eta : 1 \leftarrow \mathtt{A}^{\mathtt{O}_d(\cdot)}(d, 1^\eta)]\right| \leq neg(\eta) \end{array}$$

We note that if no oracle access is granted at all, then the above definition reduces to the standard notion of computational indistinguishability.

Our intuitive notion is that a message and its pattern should appear to be the same. We formalize this notion by saying that a message and its pattern should encode to computationally indistinguishable probability distributions. To make this formalization completely meaningful, however, we must consider what oracle (if any) the adversary can access. This will be determined by the oracles allowed by the underlying computational encryption scheme.

We will assume that the encoding mapping uses CCA-2 secure cryptography. Thus, the oracle we will use in Definition 17—to show that a message and its pattern produce indistinguishable encodings—will exactly mirror the decryption oracles of Definition 13. Those oracles will decrypt, with respect to a given public key, anything but a given "challenge" ciphertext. Our oracles will do the same. However, a message and its pattern can be thought of as possibly many different "challenge" ciphertexts under possibly many different keys. It is simple to define the keys with respect to which our oracles will decrypt:

Definition 18 *Let M be a pattern. Then $M|_{\mathcal{K}_{Pub}} = \{K \in \mathcal{K}_{Pub} : K \sqsubseteq M\}$. If S is a set of messages, then $S|_{\mathcal{K}_{Pub}} = \{K \in \mathcal{K}_{Pub} : \exists M \in S \text{ s. t. } K \sqsubseteq M\}$.*

In addition, the oracle may decrypt with respect to additional keys in some set T. (We use this additional flexibility in the proof of our main theorem.) Due to efficiency concerns, however, the set T must be finite.

It is more difficult to define the "challenge" ciphertexts which our oracle will not decrypt. Most directly, they are those encryptions which differ between $[M]_\eta^t$ and $[\![pattern_{pk}(M,T)]\!]_\eta^t$. That is, the challenge ciphertexts should be those which correspond to "blobs" in the pattern of M relative to the set of keys T. However, for convenience, we will define a larger but equivalent set of challenge ciphertexts which correspond not only to the "blobs" but all encryptions visible in M to a Dolev-Yao adversary.

Definition 19 (Visible) *Let σ be a bit-string, and τ a set of computational public keys. Then let $vis_\tau(\sigma)$ be the smallest set so that*

- *$\sigma \in vis_\tau(\sigma)$,*
- *if $\langle a, b, \text{"pair"}\rangle \in vis_\tau(\sigma)$, then $a \in vis_\tau(\sigma)$ and $b \in vis_\tau(\sigma)$,*
- *if $\langle c, k, \text{"enc"}\rangle \in vis_\tau(\sigma)$, $k \in \tau$, and k' is the secret key corresponding to k, then $\mathsf{D}(c, k') \in vis_\tau(\sigma)$, and*
- *if $\langle c, k, \text{"enc"}\rangle \in vis_\tau(\sigma)$, $\langle k', \text{"privkey"}\rangle \in vis_\tau(\sigma)$, and k' is the secret key corresponding to k, then $\mathsf{D}(c, k') \in vis_\tau(\sigma)$.*

A bit-string m is a visible element *in σ relative to τ if $m \in vis_\tau(\sigma)$.*

Intuitively, $x \in vis_\tau(\sigma)$ iff x is an encoding of X, σ is an encoding of M, τ is an encoding of T and $X \sqsubseteq pattern_{pk}(M,T)$. That is, a bit-string is a visible element of σ if the adversary can derive it from σ using only Dolev-Yao-style operations using σ and keys in τ. The set $vis_\tau(\sigma)$ contains every ciphertext which corresponds to a "blob" in $pattern_{pk}(M,T)$. However, it also contains every other ciphertext that has an corresponding analogue in $pattern_{pk}(M,T)$. The decryption oracle will not decrypt these, but this is not worrisome: the computational adversary can decrypt these "non-blobs" itself. Just as these encryptions are not "blobbed" in $pattern_{pk}(M,T)$ because the required formal private key is in T or derivable from M, the adversary can decrypt the corresponding computational ciphertext from keys in τ or derivable from σ itself. Thus, we can prohibit the decryption of this more general set without losing generality.

Now that we know the nature of our decryption oracle, we can finally define our indistinguishability property between messages and their patterns:

Definition 20 (Dolev-Yao public-key indistinguishability) *A computational encryption scheme provides* Dolev-Yao public-key indistinguishability *if, when used in $[\![\cdot]\!]_\eta^t$, for all acyclic formal messages M and finite $T \subseteq \mathcal{K}_{Pub}$:*

$$[\![M]\!]_\eta \cong_{\mathsf{O}_x^{M,T}} [\![pattern_{pk}(M,T)]\!]_\eta$$

where $\mathsf{O}_d^{T,M}(x,\mathsf{e})$ returns $\perp$ unless e is a valid public key and

- *either $\mathsf{e} \in [\![K]\!]_\eta^t$ for some $K \in T$, or*
- *$\mathsf{e} \in [\![K]\!]_\eta^t$ for some $K \in (M|_{\mathcal{K}_{Pub}} \setminus T)$ and x is not in $vis_{[\![T]\!]_\eta^t}(d)$.*

(The tape t is assumed to be consistent with that used to form the sample from $[\![pattern_{pk}(M,T)]\!]_\eta$ or $[\![M]\!]_\eta$.) In these cases, $\mathsf{O}_d^{T,M}(x,\mathsf{e})$ returns $\mathsf{D}(x,\mathsf{d})$ where d is the private key corresponding to e.

In the next section, we will show that Dolev-Yao public-key indistinguishability implies Dolev-Yao weak non-malleability. Before this, however, we show that Dolev-Yao public-key indistinguishability can be satisfied by CCA-2 security.

Theorem 3 *If the public-key encryption scheme $(\mathsf{G},\mathsf{E},\mathsf{D})$ provides indistinguishability under the chosen-ciphertext attack, $(\mathsf{G},\mathsf{E},\mathsf{D})$ provides Dolev-Yao public-key indistinguishability.*

Proof.

Suppose that the encoding mapping uses a computational encryption scheme $(\mathsf{G},\mathsf{E},\mathsf{D})$. Further, suppose that there exists a formal message M, a set of keys T and a *PPT* adversary $\mathtt{A}$ that can distinguish between a sample from $[\![M]\!]_\eta^t$ and a sample from $[\![pattern_{pk}(M,T)]\!]_\eta^t$ (given access to the oracle in Definition 20). Then $(\mathsf{G},\mathsf{E},\mathsf{D})$ does not satisfy CCA-2 security.

We prove this by hybrid argument. Since M is acyclic, we can order the key-pairs used in the parse tree of M as $K_1, K_2 \ldots K_k$ so that if $K_i \rightarrow K_j$ in the graph G_M, then $i \geq j$. That is, the deeper the key in the encryptions, the smaller the number.

We go about the hybrid argument by constructing a number of intermediate patterns between M and $\mathit{pattern}_{pk}(M, T)$. In particular, we construct patterns M_0, M_1,… M_k such that:

- $M_0 = M = \mathit{pattern}_{pk}\left(M, T \cup \left\{K_1^{-1}, K_2^{-1}, \ldots K_k^{-1}\right\}\right)$,
- $M_i = \mathit{pattern}_{pk}\left(M, T \cup \left\{K_{i-1}^{-1}, K_{i-2}^{-1}, \ldots K_k^{-1}\right\}\right)$, and
- $M_k = \mathit{pattern}_{pk}(M, T)$.

That is, between M_i and M_{i+1} we pick a key K and replace all encryptions with that key with blobs of the appropriate length.

We use this typeface for a running example. Suppose

$$M = \{\!|A|\!\}_{K_1} \; \left\{\!\left|K_1^{-1}\right|\!\right\}_{K_2} \; \{\!|B|\!\}_{K_3} \; \{\!|A\,B|\!\}_{K_2}$$

and

$$T = \{K_3, K_4\}$$

Assume for now that $\mathcal{K}_{Adv} = \emptyset$. The pattern of M is

$$\mathit{pattern}_{pk}(M, T) = \langle\!|\mathcal{M}|\!\rangle_{K_1} \; \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} \; \{\!|B|\!\}_{K_3} \; \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2}$$

By using the order on keys suggested by the notation, we can let

$$\begin{array}{rcllll}
M_0 = M & = & \{\!|A|\!\}_{K_1} & \left\{\!\left|K_1^{-1}\right|\!\right\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\
M_1 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \left\{\!\left|K_1^{-1}\right|\!\right\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\
M_2 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} & \{\!|B|\!\}_{K_3} & \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2} \\
M_3 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} & \{\!|B|\!\}_{K_3} & \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2} \\
M_4 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} & \{\!|B|\!\}_{K_3} & \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2}
\end{array}$$

We will use the hybrid argument on this table.

Now, suppose that the distributions $[\![M]\!]_\eta$ and $[\![\mathit{pattern}_{pk}(M, T)]\!]_\eta$—the top and bottoms rows of our table—are distinguishable. That is, $[\![M]\!]_\eta \not\approx_{\mathsf{0}_x^{M,T}} [\![\mathit{pattern}_{pk}(M, T)]\!]_\eta$. Then we know by a (standard) hybrid argument that two consecutive rows are also distinguishable.[4] We continue the

[4]This only follows if the number of rows in the table is polynomial in the security parameter. In this case, however, the number of rows in the table is constant with respect to η.

hybrid argument by creating a new table between the two distinguishable rows. Suppose that K_i is the key being "blobbed" between the two rows. Then there are a fixed number of encryptions being converted to "blobs". Create a row for each such encryption, so that two consecutive rows differ only in a single encryption being replaced with a blob.

For example, if the two rows are

$$M_1 = \langle\!|\mathcal{M}|\!\rangle_{K_1} \ \{\!|K_1^{-1}|\!\}_{K_2} \ \{\!|B|\!\}_{K_3} \ \{\!|A\,B|\!\}_{K_2}$$

and

$$M_2 = \langle\!|\mathcal{M}|\!\rangle_{K_1} \ \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} \ \{\!|B|\!\}_{K_3} \ \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2}$$

Then we could expand this into the table:

$$\begin{array}{rcllll} M_1 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \{\!|K_1^{-1}|\!\}_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\ M_{1.5} & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} & \{\!|B|\!\}_{K_3} & \{\!|A\,B|\!\}_{K_2} \\ M_2 & = & \langle\!|\mathcal{M}|\!\rangle_{K_1} & \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2} & \{\!|B|\!\}_{K_3} & \langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2} \end{array}$$

Two of these rows must be distinguishable.

Again, there must exist two consecutive rows R_1 and R_2 that can be distinguished. Since the rows differ only in the contents of a single encryption and every other part of the row can be created independently, distinguishing between the encoding of two rows reduces to distinguishing between two encryptions.

Let $\mathtt{A}$ be the adversary that can distinguish between the two rows, and let $E = \{\!|P|\!\}_K$ be the encryption that is being changed into $\langle\!|\mathcal{T}_P|\!\rangle_K$. Then to break the CCA-2 security of the encryption scheme we will distinguish between an encryption of m_0 and m_1 under public key $\mathtt{e}$ by:

- letting $m_0 \leftarrow [\![P]\!]_\eta^t$,
- letting $m_1 \leftarrow [\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$, and
- treating $\mathtt{e}$ as the encoding of K.

More formally:

- On input $\mathtt{e}$, select random $t \leftarrow \{0,1\}^\omega$. Then draw $p \leftarrow [\![P]\!]_\eta^t\,[\mathtt{e}/K]$, where $[\![M]\!]_\eta^t\,[x/X, y/Y, \ldots]$ is the same as $[\![M]\!]_\eta^t$ except that x is assumed to be the value for X, y the value for Y, and so on. (If X is an encryption and occurs more than once, then x is used as the value for the instance of X indicated by context. Values for the other instances are still drawn as before.) Note that because M is acyclic, we do not need to know the value $[\![K^{-1}]\!]_\eta^t$ to draw from $[\![P]\!]_\eta^t$.

Return p and $[\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$ as candidate plaintexts. (Recall that $[\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$ is a fixed string of the appropriate length, such as the all-zero string.)

- On input c, an encryption of either $[\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$ or p, sample $s \leftarrow [\![R_1]\!]_\eta^t\,[c/P, \mathsf{e}/K]$. Note that, since both t and e were selected randomly, $[\![R_1]\!]_\eta^t\,[c/P, \mathsf{e}/K]$ is the same distribution as $[\![R_1]\!]_\eta$ if c encrypts p. Similarly, $[\![R_1]\!]_\eta^t\,[c/P, \mathsf{e}/K]$ is the same distribution as $[\![R_2]\!]_\eta$ if c encrypts $[\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$. Feed $(s, 1^\eta)$ to $\mathtt{A}$.

- If $\mathtt{A}$ makes an oracle call on (x, e), we check that $\mathsf{e} = [\![K_0]\!]_\eta^t$ for some $K_0 \in M|_{\mathcal{K}_{Pub}} \cup T$. If not, we return $\perp$. If so, we decrypt or not as follows:

  - If $K_0 = K$, we check that x is not visible in x relative to $t = [\![T]\!]_\eta^t$. Since x is not visible in s relative to t, and c is visible in s relative to t, $x \neq c$. Hence, the decryption oracle D_2 in Definition 13 will happily decrypt x for us.
  - If $K_0 \neq K$, we can produce $[\![K_0^{-1}]\!]_\eta^t$ ourselves from the tape t. If $K_0 \in T$, we decrypt x with the value so produced. If $K_0 \in M|_{\mathcal{K}_{Pub}} \setminus T$, we also check to see if x is visible in s relative to t. We return $\perp$ if it is, and decrypt x if it is not.

Assume in our example that rows $M_{1.5}$ and M_2 can be distinguished by $\mathtt{A}$. *Then $P = A\,B$ and $K = K_2$. We build the two candidate ciphertexts by selecting $t \leftarrow \{0,1\}^\omega$. We then select $p \leftarrow [\![A\,B]\!]_\eta^t$, and return p, $[\![\langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle]\!]_\eta^t$ as candidate ciphertexts. When we get* c, *a value either from $[\![\langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle_{K_2}]\!]_\eta^t$ or $[\![\{\!|A\,B|\!\}_{K_2}]\!]_\eta^t$, we draw*

$$s \leftarrow [\![\langle\!|\mathcal{M}|\!\rangle_{K_1}\ \langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2}\ \{\!|B|\!\}_{K_3}\ \{\!|A\,B|\!\}_{K_2}]\!]_\eta^t\,[c/\,\{\!|A\,B|\!\}_{K_2}, \mathsf{e}/K_2]$$

Since either $s \in supp\ [\![M_{1.5}]\!]_\eta^t$ or $s \in supp\ [\![M_2]\!]_\eta^t$, the adversary $\mathtt{A}$ *will tell us which one, and this answer will tell us if* c *encrypts $[\![A\,B]\!]_\eta^t$ or $[\![\langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle]\!]_\eta^t$.*

We simulate $\mathtt{A}$ *on s: when* $\mathtt{A}$ *requests that we decrypt a string x with $[\![K_1^{-1}]\!]_\eta^t$, we make sure that it isn't* c, *$[\![B]\!]_\eta^t$, or the bit-strings in s that represent $\langle\!|\mathcal{M}|\!\rangle_{K_1}$ and $\langle\!|\mathcal{K}_{Priv}|\!\rangle_{K_2}$. If it is not these four things, we use the tape t to create the secret key and decrypt x.If* $\mathtt{A}$ *asks us to decrypt something with $[\![K_2^{-1}]\!]_\eta^t$, we check that it is not any of the four ciphertexts above. If it is not, then we send it to the decryption oracle provided to us in Definition 13, which will decrypt it for us. If* $\mathtt{A}$ *asks us to decrypt with $[\![K_3^{-1}]\!]_\eta^t$ or $[\![K_4^{-1}]\!]_\eta^t$, we create the keys from the tape and decrypt any ciphertext.*

The answer from $\mathtt{A}$ *directly corresponds to the plaintext chosen for* c, *which allows us to distinguish whether it encrypts $[\![\langle\!|\mathcal{M}\,\mathcal{M}|\!\rangle]\!]_\eta^t$ or p.*

$\mathtt{A}(s, \eta)$ will eventually return an answer that distinguishes between samples from R_1 and R_2. The answer from $\mathtt{A}$ will signify whether c encrypted p or $[\![\langle\!|\mathcal{T}_P|\!\rangle]\!]_\eta^t$. ■

We note as a corollary that the exact analogue of the Abadi-Rogaway result holds: if two messages M and N have the same pattern (with respect to some set T) then they produce indistinguishable encodings:

Corollary 4 *Suppose that M, N are two acyclic messages, $T \subseteq \mathcal{A}$ is a set of keys, and $M|_{\mathcal{K}_{Pub}} = N|_{\mathcal{K}_{Pub}}$. If $pattern_{pk}(M,T) = pattern_{pk}(N,T)$, then $[\![M]\!]_\eta \cong_{\mathtt{O}_x^{M,T}} [\![N]\!]_\eta$.*

Proof. By assumption and Theorem 20, we know that

$$[\![M]\!]_\eta \cong_{\mathtt{O}_x^{M,T}} [\![pattern_{pk}(M,T)]\!]_\eta = [\![pattern_{pk}(N,T)]\!]_\eta \cong_{\mathtt{O}_x^{N,T}} [\![N]\!]_\eta$$

Since $M|_{\mathcal{K}_{Pub}} = N|_{\mathcal{K}_{Pub}}$ and $pattern_{pk}(M,T) = pattern_{pk}(N,T)$, the oracle $\mathtt{O}_x^{M,T}$ is the same as the oracle $\mathtt{O}_x^{N,T}$. The $\cong_{\mathtt{O}_x^{M,T}}$ relation is transitive (by hybrid argument), and so the result follows. ■

We end by noting that we do not lose generality in this corollary by requiring that $M|_{\mathcal{K}_{Pub}} = N|_{\mathcal{K}_{Pub}}$. If M and N have the same pattern but have different public keys in their parse trees, then we can simply form M' by pairing with M every key in $M|_{\mathcal{K}_{Pub}} \cup N|_{\mathcal{K}_{Pub}}$, and similarly for N'. Since we add only public keys, $pattern_{pk}(M',T) = pattern_{pk}(N',T)$. However, it is now the case that $M'|_{\mathcal{K}_{Pub}} = N'|_{\mathcal{K}_{Pub}}$ and the corollary holds.

2.3 Strong Dolev-Yao Non-Malleability

Theorem 5 *Suppose that $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is a computational public-key encryption scheme that provides Dolev-Yao public-key indistinguishability. Then $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ provides both finite and infinite strong Dolev-Yao non-malleability.*

Proof.

Suppose that the theorem is false. Then there is an adversary that is able to produce a message outside the closure of its input set. In the finite case:

$$\begin{array}{ll}
\multicolumn{2}{l}{\exists\, \mathtt{A}_{PPT},\ \exists \text{finite, acyclic } S \subseteq \mathcal{A}_\eta, \exists \text{ polynomial } q, \text{ for infinitely many } \eta:} \\
\Pr[& t \leftarrow \{0,1\}^\omega \\
& s \leftarrow [\![S]\!]^t_\eta\,; \\
& m \leftarrow \mathtt{A}^{\mathtt{M}^t_\eta(\cdot), \mathtt{PbK}^t_\eta(\cdot), \mathtt{PrK}^t_\eta(\cdot), \mathtt{R}^t_\eta(\cdot)}(1^\eta, s): \\
& \exists M \in \mathcal{A}_\eta \setminus C[S]\,.m \in supp\ [\![M]\!]^t_\eta \qquad] \geq \frac{1}{q(\eta)}
\end{array}$$

In the infinite case:

$$\begin{aligned}
&\exists\, \mathtt{A}_{PPT},\ \exists\,\text{finite, acyclic } S \subseteq \mathcal{A}, \exists \text{ polynomial } q,\ \text{for infinitely many } \eta :\\
&\Pr[\quad t \leftarrow \{0,1\}^{\omega}\\
&\qquad s \leftarrow [\![S]\!]_{\eta}^{t};\\
&\qquad m, \tau \leftarrow \mathtt{A}^{\mathtt{M}_{\eta}^{t}(\cdot), \mathtt{PbK}_{\eta}^{t}(\cdot), \mathtt{PrK}_{\eta}^{t}(\cdot), \mathtt{R}_{\eta}^{t}(\cdot)}(1^{\eta}, s, t\,(S)) :\\
&\qquad \exists M \in \mathcal{A} \setminus C[S]\,.\tau = t\,(M) \ \text{ and } m \in \mathit{supp}\ [\![M]\!]_{\eta}^{t} \quad] \geq \tfrac{1}{q(\eta)}
\end{aligned}$$

In each of these cases, we will construct from the counter-example adversary a new adversary $\mathtt{A}_1$ that serves as a counter-example to Theorem 20. But first, consider the parse tree of M. Suppose that every path from the root of the parse tree to a leaf passes through an element of $C[S]$. Then it must be that the root message, M, is in $C[S]$ — a contradiction. Hence, there must be some path in the parse tree of M such that no element along that path is in $C[S]$, including the leaf M_l.

Now, consider the simple, intermediate adversary $\mathtt{A}_2$, which operates as follows:

1. It first chooses a random tape $t \leftarrow \{0,1\}^{\omega}$.

2. It then uses that tape to sample $s \leftarrow [\![S]\!]_{\eta}^{t}$.

3. It simulates the counter-example adversary $\mathtt{A}$ on input $(1^{\eta}, s)$.

4. When $\mathtt{A}$ makes an oracle query, $\mathtt{A}_2$ responds appropriately. (Because it knows the random tape t, it can compute any atomic value it wishes, including those returned by the oracles.)

5. When $\mathtt{A}$ responds with $m \in \mathit{supp}\ [\![M]\!]_{\eta}^{t}$, $\mathtt{A}_2$ uses this to produce a value $m_l \in [\![M_l]\!]_{\eta}^{t}$. That is, it progresses down the path in the parse tree of M that leads to M_l:

   - It starts with a value for $[\![M]\!]_{\eta}^{t}$, and at the root of the parse tree.
   - If the current node is a pair, $M'\,N'$, then it separates the current bit-string value into $[\![M']\!]_{\eta}^{t}$ and $[\![N']\!]_{\eta}^{t}$. It progresses down the path in the parse tree toward M_l, and keeps the value for the new node as its new current value.
   - If the current node is an encryption, $\{\![M']\!\}_K$, it uses the tape t to find the value for $[\![K^{-1}]\!]_{\eta}^{t}$. It then uses that to decrypt the current bit-string value to get $[\![M]\!]_{\eta}^{t}$, and progresses down the path in the parse tree toward M_l. (Note: we know that M_l cannot be K, since $K \in C[S]$ and we know this to not be the case for M_l.)

At the end, this adversary will have a value for $[\![M_l]\!]_{\eta}^{t}$. Now, consider what M_l might be:

- M_l cannot be a compound term, since it is a leaf of the parse tree.

- Suppose $M_l \in \mathcal{M}$. Then $M_l \in C[S]$, no matter what S is—a contradiction.

- Suppose $M_l \in \mathcal{K}_{Pub}$. Then, as mentioned above, $M_l \in C[S]$ always.

- Suppose $M_l \in \mathcal{R}$. If $M_l \in \mathcal{R}_{Adv}$ then $M_l \in C[S]$. So, we only need to worry about $M_l \in \mathcal{R} \setminus \mathcal{R}_{Adv}$. There are two cases: either M_l is in the parse tree of something in S, or it is not.

  The second case leads to a contradiction. If M_l is not in the parse tree of any element of S, then the input to the adversary is completely independent of the output. Thus, the adversary in question is able to produce a η-bit value which happens to be the encoding of a nonce M_l. If we are discussing the setting of finite strong Dolev-Yao non-malleability, then there are only η nonces, and the probability that the adversary can guess the encoding of one is bounded above by $\frac{\eta}{2^\eta}$. Thus, the adversary cannot produce a nonce not in S with non-negligible probability. If, on the other hand, we are in the infinite setting, then the adversary must have also produced a tag τ. Since $t = t\left(M\right)$, one can recover $t\left(M_l\right)$ by descending the parse tree of τ. Thus, the adversary is able to recover the encoding of one particular nonce, M_l, which has a random η-bit encoding. Again, the adversary cannot do this with non-negligible probability. Thus, in neither case can it be with non-negligible probability that M_l is in $\mathcal{R} \setminus \mathcal{R}_{Adv}$ but not in the parse tree of anything in S.

- Suppose $M_l \in \mathcal{K}_{Priv}$. Then, we proceed similar to above. If $M_l \in \mathcal{K}_{Adv}$, then $M_l \in C[S]$. If $M_l \in \mathcal{K}_{Priv} \setminus \mathcal{K}_{Adv}$ but not in the parse tree of some element of S, the adversary is able to guess a private key based on the corresponding public key, possibly encryptions using the public key, and values independent of the private key. Since we are assuming that the encryption scheme provides indistinguishability against chosen-ciphertext attacks, the probability of this must be negligible. Again, we find a contradiction.

Thus, the only possibility that occurs with non-negligible probability is that M_l is in $\mathcal{R} \setminus \mathcal{R}_{Adv}$ or in $\mathcal{K}_{Priv} \setminus \mathcal{K}_{Adv}$, and that M_l is in the parse tree of some element of S. However, it cannot be the case that M_l itself is in S, or that M_l can be produced from S only by separating pairs. (If either of those were true, then M_l would be in $C[S]$ itself, a contradiction.) Thus, M_l must only appear in S in the plaintext of encryptions.

Thus, we have an adversary $\mathtt{A}_2$ which takes an element of $[\![S]\!]_\eta^t$ and produces the plaintext to some encryption in S. Granted, $\mathtt{A}_2$ created $[\![S]\!]_\eta^t$ itself and knows every secret. Hence $\mathtt{A}_2$ does not serve as the counter-example to anything. However, a simple modification to $\mathtt{A}_2$ will serve as a counterexample to Theorem 20. Let:

$$S' = \begin{cases} S \cup S|_{\mathcal{R}} & \text{if } M_l \in \mathcal{R} \\ S \cup \left\{ \{\!|N_p|\!\}_K : K \in S|_{\mathcal{K}_{Pub}} \right\} & (\text{where } N_p \in \mathcal{R}_{Adv}) \text{ if } M_l \in \mathcal{K}_{Priv} \end{cases}$$

Then we will be able to distinguish between $[\![S']\!]_\eta$ and $\left[\!\left[pattern_{pk}\left(S', T\right)\right]\!\right]_\eta$ where T is $M|_{\mathcal{K}_{Pub}} \setminus S|_{\mathcal{K}_{Pub}}$. (Note that if M_l is a private key, then it is in neither $C[S]$ or T. Hence the encryption $\{\!|N_p|\!\}_{M_l^{-1}}$ will become $\langle\!|\mathcal{R}|\!\rangle_{M_l^{-1}}$ in $pattern_{pk}\left(M, T\right)$.)

Consider the adversary $\mathtt{A}_1$ that does the following:

1. It receives as input the value d, which is drawn either from $[\![S']\!]_\eta^t$ or from $[\![pattern_{pk}(S',T)]\!]_\eta^t$ (for some tape t). It separates d into d_S and d_{test}, where $d_S \in [\![S]\!]_\eta^t$ and either $d_{M_l} \in [\![M_l]\!]_\eta^t$ if M_l is a nonce, or $d_{M_l} \in \left[\!\!\left[\left\{N_p, \{\!|N_p|\!\}_{M_l^{-1}}\right\}\right]\!\!\right]_\eta^t$ or $\left[\!\!\left[\left\{N_p, \langle\!|\mathcal{R}|\!\rangle_{M_l^{-1}}\right\}\right]\!\!\right]_\eta^t$ if M_l is a private key.

2. It simulates $\mathtt{A}$ on $(1^\eta, d_S)$. (We will postpone consideration of any oracle calls that $\mathtt{A}$ makes for one moment.)

3. When $\mathtt{A}$ returns m (and τ if we are in the infinite case) $\mathtt{A}_1$ will attempt to extract the value $[\![M_l]\!]_\eta^t$ from m. That is, it recurses down the parse tree of M to M_l, separating pairs and decrypting encryptions, until it arrives at M_l. If we are in the infinite setting, then the tag τ will provide the path to M_l. If we are in the finite setting, then we simply go down every path until we reach the leaf or we are halted:

   - If $M = N_1\, N_2$ and $m = \langle n_1, n_2, \text{“pair”}\rangle$, then $\mathtt{A}$ continues recursively on n_1 or n_2 or both.
   - If $M = \{\!|N|\!\}_K$, $m = \langle c, k, \text{“enc”}\rangle$ and $k \in [\![K]\!]_\eta^t$, then $\mathtt{A}_1$ sends (c,k) to the decryption oracle. Will the decryption oracle decrypt? If not, we halt. But along the path to M_l, it must. There are two cases:
     – By definition, $K \in M|_{\mathcal{K}_{Pub}}$. If $K \notin S|_{\mathcal{K}_{Pub}}$ also, then $K \in T$. Hence, the oracle of Defintion 20 will decrypt $\mathtt{c}$.
     – If $K \in S|_{\mathcal{K}_{Pub}}$, then $K \in S|_{\mathcal{K}_{Pub}} \backslash (M|_{\mathcal{K}_{Pub}} \setminus S|_{\mathcal{K}_{Pub}})$. But $S|_{\mathcal{K}_{Pub}} \backslash (M|_{\mathcal{K}_{Pub}} \setminus S|_{\mathcal{K}_{Pub}}) = S|_{\mathcal{K}_{Pub}} \setminus T$. Hence, the decryption oracle of Definition 20 will decrypt $\mathtt{c}$ if $\mathtt{c}$ is not in $vis_{[\![T]\!]_\eta^t}(d)$. However, could $\mathtt{c}$ be visible in d with respect to $[\![T]\!]_\eta^t$? If it is, then by the definition of visibility, $\{\!|N|\!\}_K \sqsubseteq pattern_{pk}(S,T)$. In this case, however, $T = M|_{\mathcal{K}_{Pub}} \setminus S|_{\mathcal{K}_{Pub}}$, and so contains no keys in the parse tree of S. Allowing the adversary to decrypt with respect to T does not give it more information about S. Hence, $pattern_{pk}(S,T) = pattern_{pk}(S)$. Thus, if $\{\!|N|\!\}_K \sqsubseteq pattern_{pk}(S,T)$ then $\{\!|N|\!\}_K \sqsubseteq pattern_{pk}(S)$ and so by Theorem 2 it must be that $\{\!|N|\!\}_K \in C[S]$. However, this contradicts the assumption that no node on the path from M to M_l is in $C[S]$, and so $\mathtt{c}$ cannot be visible in d. Hence, the decryption oracle of Definition 20 will decrypt it.

   Thus, the decryption oracle will always return p, the plaintext of $\mathtt{c}$. $\mathtt{A}_1$ then moves down the parse tree to the node for N and recursively applies this process to p.

4. If we fail to reach a leaf M_l which is not in the closure, then $\mathtt{A}$ immediately stops and outputs 0. Otherwise, $\mathtt{A}_1$ will acquire a value m_l which may be the encoding of M_l. $\mathtt{A}_1$ tests this using the string d_{test}, which it reserved at the beginning. If M_l is a nonce, then d_{test} will contain the value for M_l; $\mathtt{A}$ can simply test that m_l is a substring of d_{test}. If M_l is a private key, then

d_{test} contains an encryption of a known plaintext $[\![N_p]\!]^t_\eta$ under the corresponding public key. $\mathtt{A}_1$ simply decrypts each encryption with m_l. One of the results should be the same as $[\![N_p]\!]^t_\eta$. If these tests are satisfied, then $\mathtt{A}_1$ outputs 1. Otherwise, it outputs 0.

$\mathtt{A}_1$ will return 1 whenever $\mathtt{A}$ produces an element of *supp* $[\![M]\!]^t_\eta$. Hence, $\mathtt{A}_1$ will return 1 with probability at least $\frac{1}{q(\eta)}$ given that d is in fact drawn from $[\![S]\!]_\eta$. If, on the other hand, d is drawn from $[\![pattern_{pk}(M,T)]\!]_\eta$, then $\mathtt{A}$ cannot have a non-negligible chance of producing a $m \in supp\ [\![M]\!]^t_\eta$ for any $M \in \mathcal{A} \setminus C[S]$ (or $\mathcal{A}_\eta \setminus C[S]$ in the finite case). Since $M_l \notin C[S]$, it cannot be that $M_l \sqsubseteq pattern_{pk}(S) = pattern_{pk}(S,T)$.

- If M_l is a nonce, then this implies that the sample d will be entirely independent of the actual value for $[\![M_l]\!]^t_\eta$.
- If M_l is a private key, on the other hand, then d may include encryptions made using the public key $[\![M_l^{-1}]\!]^t_\eta$. But the encryption provides indistinguishability against chosen-ciphertext attack, so it is infeasible to recover a private key using only encryptions under the corresponding public key. Since d is otherwise independent of $[\![M_l]\!]^t_\eta$, $\mathtt{A}$ cannot have a non-negligible chance of recovering $[\![M_l]\!]^t_\eta$.

Thus, the probability that $\mathtt{A}_1$ will return 1 given that d is sampled from $[\![pattern_{pk}(S,T)]\!]^t_\eta$ must be bounded above by either:

- The probability of guessing some nonce or private key, in the finite setting, or
- The probability of guessing the particular nonce or private key indicated by the tag, in the infinite setting.

In both of these cases, the probability of a successful guess must be negligible.

Hence, if $\mathtt{A}$ has a non-negligible chance of constructing $m \in supp\ [\![M]\!]^t_\eta$ from a sample from $[\![S]\!]^t_\eta$, for any $M \notin C[S]$, then $\mathtt{A}_1$ has a non-negligible chance of distinguishing $[\![S']\!]_\eta$ from $[\![pattern_{pk}(S',T)]\!]_\eta$, a contradiction of Theorem 20.

There remains only one last complication: $\mathtt{A}$ has access to oracles while operating. In particular, $\mathtt{A}$ can request any public key, any private key in $\mathcal{K}_{Adv}$, any identifier, and any nonce in $\mathcal{R}_{Adv}$. How does $\mathtt{A}_1$ respond to these oracle calls when it simulates $\mathtt{A}$?

The answer is that we slightly modify the set S' to include the information needed to respond. In particular, let $S|_{\mathcal{K}_{Pub}}$ and $S|_{\mathcal{R}_{Adv}}$ be defined analogously to $S|_{\mathcal{K}_{Pub}}$.Then the set S' is will actually be

$$S' = \begin{cases} S \cup S|_{\mathcal{K}_{Pub}} \cup S|_{\mathcal{K}_{Adv}} \cup S|_{\mathcal{R}_{Adv}} \cup \{M_l\} & \text{if } M_l \in \mathcal{R} \\ S \cup S|_{\mathcal{K}_{Pub}} \cup S|_{\mathcal{K}_{Adv}} \cup S|_{\mathcal{R}_{Adv}} \cup \left\{N_p, \{\!|N_p|\!\}_{M_l^{-1}}\right\} & \text{if } M_l \in \mathcal{K}_{Priv} \end{cases}$$

When $\mathtt{A}_1$ receives the input d it strips off d_S as before and simulates $\mathtt{A}$. When $\mathtt{A}$ makes an oracle call, however, $\mathtt{A}_1$ can respond:

- If the oracle is being asked for an identifier, $\mathtt{A}_1$ computes the representation of that identifier. (As mentioned before, we assume that the encoding of identifiers is efficiently computable.)
- If the oracle is being called on an ingredient of S, then the additional information in s contains the needed bit-string.
- Otherwise, the needed value is a random variable independent of d. $\mathtt{A}_1$ can sample from the relevant distribution to produce an indistinguishable value. It then stores the value for future use (and if the value is a key, the corresponding secret or public key also), and returns it.

Since the formal messages we added to S' are already in $C[S]$, they do not change the pattern of the original S. Hence, adding them to S' does not change the distribution of d_S, and $\mathtt{A}$ will progress as before. ■

Thus, both of our stronger formalizations of the Dolev-Yao assumption can be satisfied by standard, known definitions of computational security. Thus, the results of [30] are no longer hindered by the criticisms of plaintext-aware encryption. In the next chapter, however, we address these criticisms directly by proposing a new definition (and implementation) of plaintext awareness itself.

Chapter 3

Plaintext Awareness via Key Registration

As stated above, an encryption scheme is *plaintext-aware* if, whenever an adversary creates a ciphertext, it must "know" its corresponding plaintext. However, existing definitions of plaintext-awareness have been rightfully criticized for their reliance upon the existence and the trustworthiness of the random oracle. In this chapter, we discuss our work in this area [31] which demonstrates that this criticism applies only to the original definitions, not the underlying notion. In particular, we put forth two main contributions: a new definition and an implementation.

1. As opposed to previous definitions, our definition of plaintext-aware encryption does not use the random oracle. We still need a trusted third party, but our party is much more natural (being already used in practice), much less trusted, and is used only once rather than at every encryption.

2. Our implementation is a self-referential variation on an earlier, well-known encryption scheme due to Sahai [59]. In particular, our variation will require the sender to encrypt messages in its own keys as well as those of the intended sender. Strong cryptographic primitives called *zero-knowledge proofs* will prove to the receiver that the sender performed this action and that the sender knows his or her own private keys. Thus, the receiver can know that the sender is able to decrypt any ciphertext that it creates.

3.1 A New Definition of Plaintext Awareness

Our new definition requires a particular model for public-key encryption: All users will have either or both of a *receiving* key-pair and a *sending* key-pair. Furthermore users are assumed to registered their public sending keys with a trusted *registration authority*. Lastly, we make a new assumption about the adversary: that it is *history-preserving*, meaning that it records and never erases its input and internal coin-flips.

However, these are not radical changes. Firstly, users already have two key-pairs: one for signing messages (which corresponds to the sending key-pair) and one to decrypt messages sent to them (the receiving key-pair). Furthermore, a trusted registration authority is essentially already implicit in any actual implementation of public-key encryption. Such implementations enforce an association between users and public keys by requiring that users register their public key with a *certification authority*. These authorities verify the identity of the applicant and that the applicant knows the corresponding secret key. It is natural, therefore, to assume the existence of a trusted third party for key registration. Lastly, it is not unreasonable to expect the adversary to record its input and coin-flips: we do not weaken the abilities of the adversary by prohibiting erasures.

Our definition of plaintext awareness requires that the adversary know the plaintext to *any* ciphertext it creates, provided the (apparent) sender has registered its sending key with an honest registration authority. In particular, there should exist an extractor which can, given that a successful registration had previously occurred, predict the output of the receiver's decryption algorithm. However, the addition of a trusted authority should only strengthen the security of the scheme, not weaken it. Our definitions, therefore, require that the scheme should remain CCA-2 secure (i.e., the most secure as is possible without a trusted third party) even if the registration authority is itself corrupt.

Our new definitions serve as yet another example of one of the grand paradigms of cryptography: if one wishes a protocol or primitive P to have some property, it can often be achieved by adding a set-up phase that provides P. For example, one can achieve secrecy of communication by engaging in a set-up phase in which a secret key is exchanged. Also, authentication in the retrieval stage for A's public key later allows the authentication of messages from A.

In this chapter, we wish to add to public-key encryption the property of plaintext-awareness, or *knowledge* of the plaintext. Therefore, our definitions add a set-up phase to asymmetric encryption, and our implementation will use during this phase a protocol that achieves knowledge. In particular, we will use an (interactive) proof of knowledge protocol at set-up to achieve (non-interactive) plaintext knowledge for later encryptions.

3.1.1 Preliminaries

We say that an algorithm or interactive TM A is *history-preserving* if it "never forgets" anything. As soon as it flips a coin or receives an input or a message, A writes it on a separate history tape that is write-only and whose head always moves from left to right. The history tape's content coincides with A's internal configuration before A executes any step.

If A is an history-preserving algorithm and it A appears more than once in a piece of GMR notation (e.g, $\Pr[\ldots; a \leftarrow A(x); \ldots; b \leftarrow A(y); \ldots : p(\cdots, a, b, \cdots)]$) then the history and state of A is preserved from the end of one "use" to the beginning of the next.

The notation $h \xleftarrow{H} A$ indicates that h is the current content of A's history tape.

We assume the adversary to be an efficient (probabilistic polynomial-time) history-preserving algorithm (interactive TM).

Following [26], we consider a two-party protocol as a pair, (A, B), of interactive Turing machines. By convention, A takes input (x, r_A) and B takes input (y, r_B) where x and y are arbitrary and r_A and r_B are random tapes. On these inputs, protocol (A, B) computes in a sequence of rounds, alternating between A-rounds and B-rounds. In an A-round only A is active and sends a message (i.e., a string) that will become an available input to B in the next B-round. (Likewise for B-rounds.) A computation of (A, B) ends in a B-round in which B sends the empty message and both A and B compute a private output.

Letting E be an execution of protocol (A, B) on input (x, y) and random input (r_A, r_B), we make the following definitions:

- The *transcript* of E consists of the sequence of messages exchanged by A and B, and is denoted by $\mathsf{TRANS}^{A,B}(x, r_A|y, r_B)$;
- The *view of* A in execution E consists of the triplet (x, r_A, t), where t is E's transcript, and is denoted by $\mathsf{VIEW}_A^{A,B}(x, r_A|y, r_B)$;
- The *view of* B consists of the triplet (y, r_B, t), where t is E's transcript, and is denoted by $\mathsf{VIEW}_B^{A,B}(x, r_A|y, r_B)$;
- If E is an execution of (A, B) on inputs (x, r_A) and (y, r_B) then the *output of A in E*, denoted $\mathsf{OUT}_A^{A,B}(x, r_A|y, r_B)$, consists of the string z output by A after the last round. Similarly, $\mathsf{OUT}_B^{A,B}(x, r_A|y, r_B)$ is the output of B in the same execution.
- We also define the random distribution $\mathsf{OUT}_A^{A,B}(x, \cdot|y, \cdot)$ to be $\mathsf{OUT}_A^{A,B}(x, r_A|y, r_B)$ where r_A and r_B are selected randomly, and similarly for $\mathsf{OUT}_B^{A,B}(x, \cdot|y, \cdot)$.

We say that an execution of a protocol (A, B) has *security parameter* η if the private input of A is of the form $(1^\eta, x')$ and the private (non-random) input of B is of the form $(1^\eta, y')$.

3.1.2 Formal Definitions

A *registration-based plaintext-aware encryption scheme* consists of a registration protocol (RU, RA)and a public-key encryption scheme (G, E, D), where:

- $(\mathsf{RU}, \mathsf{RA})$ is the two-party protocol in which the sender keys are generated and registered. The registration authority RA should output e_s, the public key of the sender. The registering user RU output both e_s and d_s, the public and private keys for the sender. After a successful run of the protocol, one can imagine the public sending key is inserted in a public file or that the user is given a certificate, but the precise mechanism of publication is irrelevant here. What is crucial, however, is that the registration protocol be a *secure atomic operation.* That is, we can think of it as being run one user at a time, in person, and from beginning to end.[1]

  It is worth noting that either RU or RA may reject in the registration protocol (presumably when the other party is dishonest), in which case we assume the output is $\perp$. For ease in the definitions, we assume that if $\perp$ is any input to either E or D, the output will also be $\perp$ or $(\perp, \perp)$ as appropriate.

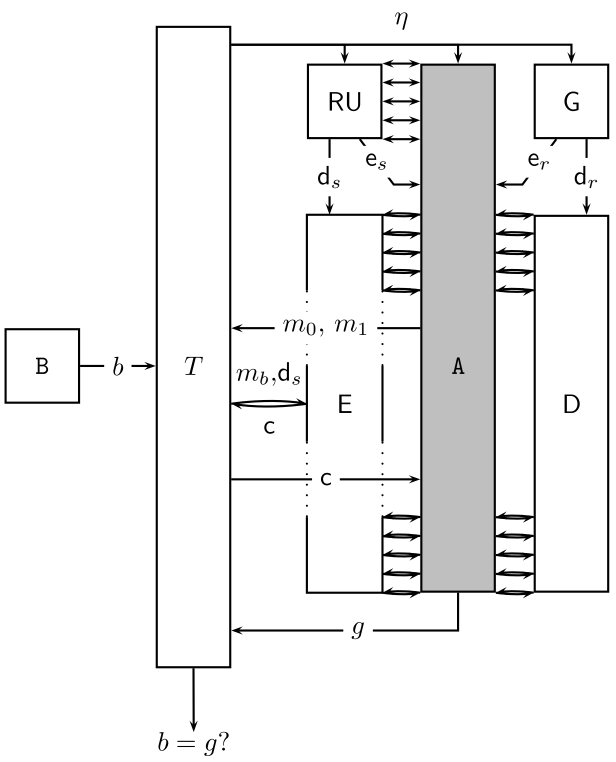
- $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is a public-key encryption scheme, where:
  - $\mathsf{G}(1^\eta)$ produces $(\mathsf{e}_r, \mathsf{d}_r)$, a key pair for the receiver,
  - $\mathsf{E}(m, \mathsf{e}_r, \mathsf{d}_s)$ produces a ciphertext c, where m is the message to encrypt, e_r is the receiver's public key, and d_s is the sender's private key. (We assume without loss of generality that a private key also contains the corresponding public key.) The ciphertext c is assumed to explicitly indicate which public keys were used in its creation.
  - $\mathsf{D}(c, \mathsf{d}_r, \mathsf{e}_s)$ produces m, a message, where c is the ciphertext to decrypt, d_r is the receiver's private key, and e_s is the sender's public key. Note that this algorithm takes both the receiver's private receiving key and the sender's public sending key. If the ciphertext is invalid, the output is $\perp$.

The protocol $(\mathsf{RU}, \mathsf{RA})$ and the encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ must satisfy the following conditions:

Registration Completeness. The key registration protocol between an honest registrant and an honest registration authority will almost always be successful. Furthermore, the user and the authority will agree on the public key.

$$\begin{array}{lll}\Pr[& r_1 \leftarrow \{0,1\}^*; r_2 \leftarrow \{0,1\}^*; & \\ & (\mathsf{e}_s, \mathsf{d}_s) \leftarrow \mathsf{OUT}_{\mathsf{RU}}^{\mathsf{RU},\mathsf{RA}}(1^\eta, r_1 | 1^\eta, r_2); & \\ & \mathsf{e}'_s \leftarrow \mathsf{OUT}_{\mathsf{RA}}^{\mathsf{RU},\mathsf{RA}}(1^\eta, r_1 | 1^\eta, r_2): & \\ & \mathsf{e}_s = \mathsf{e}'_s \neq \perp &] \geq 1 - \mathit{neg}(\eta)\end{array}$$

[1]Without this assumption, we would have to worry about man-in-the-middle, concurrency and other types of attacks which will obscure both the definitional and implementation aspects of our model.



As in the chosen-ciphertext game, the adversary cannot send the challenge ciphertext c to the decryption oracle D. (Not shown: transmission of d_r from D to T.)

Figure 3-1: The honest security game

Encryption Completeness. If an honest sender encrypts a message m into a ciphertext c, then the honest recipient will almost always decrypt c into m.

$$\begin{aligned}
&\forall m \in \{0,1\}^\eta \\
&\Pr[\quad (\mathsf{e}_s, \mathsf{d}_s) \leftarrow \mathsf{OUT}^{\mathsf{RU},\mathsf{RA}}_{\mathsf{RU}}(1^\eta, \cdot|1^\eta, \cdot)\,; \\
&\qquad (\mathsf{e}_r, \mathsf{d}_r) \leftarrow \mathsf{G}(1^\eta); \\
&\qquad \mathsf{c} \leftarrow \mathsf{E}(m, \mathsf{e}_r, \mathsf{d}_s); \\
&\qquad g \leftarrow \mathsf{D}(c, \mathsf{d}_r, \mathsf{e}_s) : \\
&\qquad g = m \qquad\qquad\qquad\qquad] \geq 1 - neg(\eta)
\end{aligned}$$

Honest Security. If recipient and sender are honest, then the encryption is adaptively chosen-ciphertext secure even if the adversary controls the registration authority. Note that the definition of chosen-ciphertext security use here differs from Definition 13 in two ways. (Contrast the original chosen-ciphertext "game" of Figure 2-1 with the new honest-security "game" of the key-registration setting in Figure 3-1.) First, the encryption and decryption algorithms now use keys of both sender and receiver. More importantly, however, the adversary now also has access to an encryption oracle.

Definition 13 allows the adversary to have arbitrary ciphertexts decrypted by a decryption oracle. The corresponding encryption oracle was not needed: if the encryption algorithm used only public keys, the adversary can encrypt on its own. Now that encryption may require the sender's private key, however, we need to provide access to an encryption oracle explicitly:

$$\begin{array}{ll}
\multicolumn{2}{l}{\forall \text{ oracle-calling adversaries } \mathtt{A}} \\
\Pr[& (\mathsf{d}_r, \mathsf{e}_r) \leftarrow \mathsf{G}(1^\eta); \\
& (\mathsf{e}_s, \mathsf{d}_s) \leftarrow \mathsf{OUT}_{\mathsf{RU}}^{\mathsf{RU},\mathtt{A}}(1^\eta, \cdot | 1^\eta, \cdot)\,; \\
& m_0, m_1 \leftarrow \mathtt{A}^{\mathsf{E}(\cdot,\cdot,\mathsf{d}_s),\mathsf{D}(\cdot,\mathsf{d}_r,\cdot)}(\mathsf{e}_r, \mathsf{e}_s); \\
& b \leftarrow \{0,1\}\,; \\
& \mathsf{c} \leftarrow \mathsf{E}(m_b, \mathsf{e}_r, \mathsf{d}_s); \\
& g \leftarrow \mathtt{A}^{\mathsf{E}(\cdot,\cdot,\mathsf{d}_s),\mathsf{D}(\cdot,\mathsf{d}_r,\cdot)-\{c,\mathsf{e}_s\}}(c) : \\
& b = g \qquad\qquad\qquad\qquad] \leq \frac{1}{2} + neg(\eta)
\end{array}$$

where

- m_0 and m_1 have the same length,
- $\mathsf{E}(\cdot,\cdot,\mathsf{d}_s)$ is the oracle that returns $\mathsf{E}(m', \mathsf{e}'_r, \mathsf{d}_s)$ on input (m', e'_r).
- $\mathsf{D}(\cdot,\mathsf{d}_r,\cdot)$ is the oracle that returns $\mathsf{D}(m', \mathsf{d}_r, \mathsf{e}'_s)$ on input (m', e'_e).
- $\mathsf{D}(\cdot,\mathsf{d}_r,\cdot) - \{c, \mathsf{e}_s\}$ is the oracle that on input (c', e'_s) returns $\mathsf{D}(c', \mathsf{d}_r, \mathsf{e}'_s)$ if $c' \neq c$ or $\mathsf{e}'_s \neq \mathsf{e}_s$, and returns $\perp$ otherwise.

If the input to any of these oracles includes $\perp$, then the output will also be $\perp$. Also, recall that the adversary is assumed to be history-preserving, so that it 'remembers' e_s and e_r from the first invocation.

Plaintext Awareness. If the registration authority is honest and player `X` (either the adversary or an honest player) registers a key, then the adversary can decrypt any string it sends to an honest participant ostensibly encrypted by `X`. More specifically, there exists an extractor that can determine the result of the recipient's decryption algorithm without access to the recipient's private key. As mentioned in Section 1.6, the ciphertext alone cannot contain enough information to enable this extraction without violating the semantic (and hence chosen-ciphertext) security of the scheme. Thus, the extractor needs access to some additional information. In the original definition of plaintext-awareness, this additional information was a list of the queries made by the adversary to the random oracle. These queries presented an opportunity to see the internal state of the adversary. Since this opportunity is no longer present, we will use as our additional information the adversary's state directly—as represented by its history tape.

In our implementation, the extractor will use this history tape to simulate the adversary. That is, the extractor is assumed to have the adversary's code 'hard-wired' into it. The order of quantifiers below allows this: the extractor can be chosen after the adversary. However, this is not necessary: the extractor can be chosen before the adversary if the adversary provides its own code as well as its internal history tape. That is, our implementation will satisfy this variation as well. However, we believe that the weaker form below to be more appropriate as the "official" definition of plaintext-awareness.

However, in its effort to create an "undecipherable" challenge plaintext the adversary will have access to additional information of its own. It will have access to a decryption oracle, as above. One might also expect the adversary to have access to an encryption oracle, and in a way it will. However, a mere encryption oracle is not sufficient. To model realistic scenarios in which our encryption scheme may be used, we must consider the fact that the adversary may (as a result of higher-level protocols) gain ciphertexts to which it does not know the plaintext. It may be counter-intuitive to think that these might help the adversary in any way, but recall that the goal of the adversary is to create a ciphertext whose plaintext is unknown to it. One must consider the possibility that the adversary cannot do this on its own but can do so by manipulating a "valid" ciphertext (with an unknown plaintext) from an external source.

For this reason, it is insufficient to give the adversary access only to decryption and encryption oracles. The decryption oracle does not provide new ciphertexts, and the encryption oracle is insufficient for two reasons:

- If the adversary merely had access to an encryption oracle, then it must "know" the plaintext to all so-produced ciphertexts.

- Even if the above objection could be nullified (e.g., by allowing erasures) it still only allows the adversary access to ciphertexts whose plaintexts it can generate. Suppose, for example, that a higher-level protocol gives the adversary both a large graph and an encryption of a Hamiltonian circuit. A mere encryption oracle would not be sufficient to simulate this scenario.

We represent the possible activity of a higher-level protocol as an "ally" PPT oracle `L`. When activated, `L` examines the history of the adversary. It then generates any plaintext and public receiver key that it pleases. This plaintext is then encrypted with the appropriate sending key, and the ciphertext is given to the adversary. Note that this ally is a generalization of a simple encryption oracle: one can imagine an ally that simply outputs as message and key the last two elements of the adversary's history. Hence, there is no need to give access to an encryption oracle explicitly.

To summarize (see Figure 3-2) the definition of plaintext awareness requires that there exist an extractor that can produce the plaintexts to adversarily-chosen ciphertexts, given only the ciphertext and the adversary's history. The adversary has access to a decryption algorithm and an arbitrary

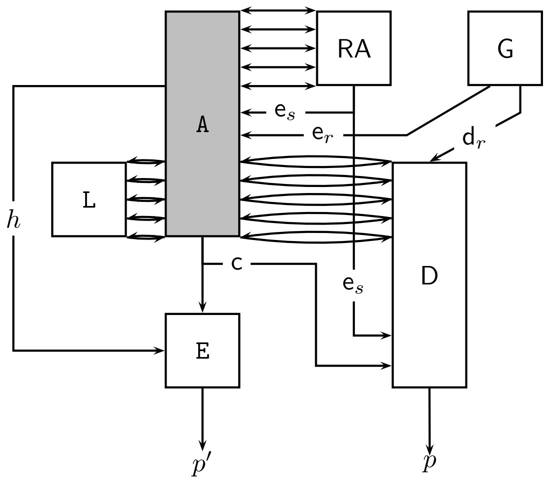


Figure 3-2: The game of plaintext awareness via key registration

ally oracle which has its own private input. The challenge ciphertext given to the extractor cannot be the result of a call to the ally. The extractor can depend on the choice of adversary, but must then work for all allies:

$$\begin{aligned}
&\forall \text{ adversaries } \mathtt{A}, \forall\, \mathtt{X} \in \{\mathtt{A}, \mathsf{RU}\}, \exists \text{ efficient algorithm } \mathtt{E_X}, \forall \text{ PPT allies } \mathtt{L} \\
&Pr[\quad (\mathsf{e}_r, \mathsf{d}_r) \leftarrow \mathsf{G}(1^\eta); \\
&\qquad \mathsf{e}_{\mathtt{X}} \leftarrow \mathsf{OUT}_{\mathsf{RA}}^{\mathtt{X},\mathsf{RA}}(\mathsf{e}_r, \cdot | 1^\eta, \cdot); \\
&\qquad h \xleftarrow{H} \mathtt{A}; \\
&\qquad c \leftarrow \mathtt{A}^{\mathtt{L}'_{\mathsf{d_X}}(\cdot), \mathsf{D}(\cdot, \mathsf{d}_r, \cdot)}(\mathsf{e}_{\mathtt{X}}, \mathsf{e}_r): \\
&\qquad \mathtt{E_X}(h, c, \mathsf{e}_r, \mathsf{e}_{\mathtt{X}}) = \mathsf{D}(c, \mathsf{d}_r, \mathsf{e}_{\mathtt{X}}) \text{ given that } c \notin \mathtt{L}_c) \quad] \geq 1 - neg(\eta)
\end{aligned}$$

where the oracle $\mathtt{L}'_{\mathsf{d_X}}$ operates as follows: on any input, run $\mathtt{L}$ on the history h' of the adversary at the time of the call to $\mathtt{L}'$. When $\mathtt{L}$ produces an output $(m, \mathsf{e}'_r, \mathsf{d}'_s)$, run $\mathsf{E}(m, \mathsf{e}'_r, \mathsf{d}'_s)$ to produce the output c. Add c to the (initially empty) set $\mathtt{L}_c$ and also return c to the adversary.

(Note that if $\mathtt{X} = \mathsf{RU}$, then it expects its input to the registration protocol to be 1^η and not e_r. Hence, we assume that if RU finds input e_r that it extracts 1^η from it and proceeds as normal. Also note that if the sender key is registered by an honest participant RU then h, the history of the adversary, will be empty.)

Remarks. Note that these definitions do not guarantee anonymity of the sender. That is, senders must register their keys, and so it might be that they can no longer send messages without their name attached in some way. We note three things with respect to this.

1. If plaintext-awareness is not required, a sender may simply use an unregistered key. Encryption will still be secure against the chosen-ciphertext attack.

2. Each registered key does not necessarily represent a sender but rather one incarnation of a

sender. Senders may register many keys in order to bolster their anonymity.

3. In our implementation, registration uses only the values of keys and not the identity of the registrant. Thus, the registration process could remain anonymous. It will still be possible to trace multiple messages back to the same sender, but the identity of the sender will not be revealed.

We choose to regard the possibility of sender authentication as an opportunity rather than a drawback, and use it in an essential way in our implementation.

3.2 Cryptographic Building-Blocks

The two implementations that follow use a wide variety of powerful cryptographic primitives. Semantically-secure encryption for public-key encryption has already been introduced (Definition 3).

We will also use digital signatures:

Definition 21 (Digital Signature Scheme) *A* digital signature scheme *is a triple* $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$ *where:*

- G_{sig} *is the (probabilistic) key generator, which generates pairs* (s, v) *of signature generation keys* (s) *and verification keys* (v)*,*
- S_{sig} *is the (probabilistic) signing algorithm, which takes an arbitrary message and a signature verification key and returns a signature, and*
- V_{sig} *is the verification algorithm, which takes a message, a signature, and a verification key to return a single bit;*

such that for all security parameters η *and messages* m*,*

$$\Pr\left[(\mathsf{s},\mathsf{v}) \leftarrow \mathsf{G}_{sig}(1^\eta); \mathsf{sig} \leftarrow \mathsf{S}_{sig}(m, \mathsf{s}) : \mathsf{V}_{sig}(m, \mathsf{sig}, \mathsf{v}) = 1\right] = 1$$

Digital signatures are the dual to public-key encryption: the signature does not hide the message in any way, and can be verified by anyone. However, only the entity that should be able to create a valid signature (one that verifies against a given message and verification key) should be the one who knows the signing key. In particular, an adversary (who does not know the signature verification key) should not be able to find a signature that verifies against a given message and verification key, even if it can choose the message itself [27]:

Definition 22 (Security for digital signatures) *A digital signature scheme is* secure against the chosen-message attack *if*

$$\begin{array}{ll} & \forall\, PPT \;\; adversaries\; \mathtt{A} : \\ \Pr[& (\mathsf{s},\mathsf{v}) \leftarrow \mathsf{G}_{sig}(1^\eta); \\ & (m,\mathsf{sig}) \leftarrow \mathtt{A}^{\mathsf{S}_{sig}(\cdot,\mathsf{s})}(1^\eta,\mathsf{v}) : \\ & \mathtt{V}(m,\mathsf{sig},\mathsf{v}) = 1 \qquad\qquad] \leq neg(\eta) \end{array}$$

It is required that the m produced by the adversary not have previously been given as input to the oracle.

Note that this definition of security could allow the adversary to transform one valid signature sig of a message m into another valid signature sig' of that same message. We will wish to prohibit this, and so consider the special case where each message has a single valid signature relative to a given verification key [28, 41, 22, 44]:

Definition 23 *A digital signature scheme $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$ provides* unique signatures *if S_{sig} is deterministic.*

We will also use a series of powerful tools based on computational *proofs*, where a proof scheme is a game between two interacting machines, one playing the *prover* and one playing the *verifier*. Suppose that L is a language in $\mathcal{NP}$ with witness relation L_R. The prover wishes to convince the verifier of a *theorem*: that a given l is in L. The real prover should be able to do so if the theorem is true and the prover has a witness for l (i.e. an element of $L_R(l)$) but no prover (even malicious ones) should be able to do so otherwise:

Definition 24 (Interactive proof system) *A pair of interactive PPT Turing machines, a prover $\mathtt{P}$ and a verifier $\mathtt{V}$, are an interactive proof system for a language $L \in \mathcal{NP}$ with witness relation L_R if:*

1. *Completeness: If $l \in L$ and the honest prover has a valid witness, then it can produce a proof which is acceptable to the verifier (except with negligible probability):*

$$\Pr\left[b \leftarrow \mathsf{OUT}_{\mathtt{V}}^{\mathtt{P},\mathtt{V}}\left(\langle l, R(l)\rangle, \cdot | l, \cdot\right) : b = 1\right] \geq 1 - neg(|l|)$$

2. *Soundness: If $l \notin L$ then no prover (even a malicious one that wishes to "fool" the verifier) can produce a proof which is acceptable to the verifier (except with negligible probability): $\mathtt{P}'$,*

$$\Pr\left[b \leftarrow \mathsf{OUT}_{\mathtt{V}}^{\mathtt{P}',\mathtt{V}}\left(l, \cdot | l, \cdot\right) : b = 1\right] \leq neg(|l|)$$

The above definition is satisfied by the trivial protocol where the prover simply sends the witness to the verifier. In our application of such protocols, we will require in addition that the proof system be *zero-knowledge* [8, 24, 26] meaning that the verifier learns nothing from the proof other than the truth of the theorem. That is, we wish that anything a (possibly malicious) verifier could do after the proof it could have also accomplished before. We formalize this by requiring the existence of a *simulator* that can create valid-seeming proofs for a given $l \in L$ but without the witness itself. Then, if the malicious verifier can perform some action after the proof, it could have done so without the proof by running the simulator and continuing as if the proof had occurred.

Definition 25 (Zero-knowledge (ZK) proof system) *A pair of interactive PPT Turing machines, a prover* $\mathtt{P}$ *and a verifier* $\mathtt{V}$*, are a* zero-knowledge *interactive proof system for a language* $L \in \mathcal{NP}$ *with witness relation* L_R *if they are an interactive proof system for* L *and for every malicious verifier* $\mathtt{V}'$ *there exists a probabilistic polynomial-time simulator* $\mathtt{S}$ *such that for every* $l \in L$ *and* $w \in L_R(l)$*:*

$$\left\{\langle l, t\rangle : t \leftarrow \mathsf{TRANS}^{\mathtt{P},\mathtt{V}'}\left(\langle l, w\rangle, \cdot|l, \cdot\right)\right\} \cong \{\langle l, t\rangle : t \leftarrow \mathtt{S}(l)\}$$

A proof of knowledge [8] is a proof system where the prover proves knowledge of a witness. At first glance, it would seem that the definition of a proof system captures this. However, there may be languages in $\mathcal{NP}$ where it is easy to decide membership but hard to find witnesses. In particular, one could imagine public-key encryption schemes where public keys are all of a certain simple form. Thus, it is easy to decide if a given string is in the language of public keys. However, this is quite different from asserting that one knows the corresponding secret key. For example, one may consider the language $\mathtt{COMP}$ of composite natural numbers, where the witness for a $x \in \mathtt{COMP}$ is its factorization. Although one can efficiently recognize elements of $\mathtt{COMP}$ [5], it is widely assumed to be hard to compute x's witness from x alone.[2]

Also, a formal definition makes the concept of algorithm's "knowledge" explicit: an algorithm "knows" a value if it is possible to extract that value from the algorithm. That is, a proof scheme for $L \in NP$ is also a proof of knowledge for the witness relation if there exists an *extractor* that can produce a witness to a given theorem by repeated interaction with the prover. Specifically, suppose that a (possibly malicious) prover can produce with probability p a valid-seeming proof for a given theorem. Then the extractor can produce a witness to that theorem with probability close to p. To do so, extractor can "reset" the prover into its original, pre-proof state, and can see and take part in multiple (interactive) proofs of the given theorem. (Note that the prover uses the same witness for each proof.)

Definition 26 (Proof of knowledge system) *A pair of interactive PPT Turing machines, a*

[2]Indeed, the security of the RSA scheme [58], where public keys are in $\mathtt{COMP}$ and witnesses are private keys, depends on this assumption.

prover $\mathtt{P}$ *and a verifier* $\mathtt{V}$*, are a* proof-of-knowledge system *for a language* $L \in \mathcal{NP}$ *with witness relation* L_R *if* $(\mathtt{P}, \mathtt{V})$ *are an interactive proof system for* L *and there exists a oracle-calling probabilistic polynomial-time machine* $\mathtt{E}$ *such that for every malicious prover* $\mathtt{P}'$*, every* $l \in L$*, every* $y \in \{0,1\}^*$ *(which may or may not be a witness to* l*), and random tape* $r \in \{0,1\}^*$*:*

$$\Pr\left[w \leftarrow \mathsf{OUT}_{\mathtt{E}}^{\mathtt{P}',\mathtt{E}}((l,y), r|l,\cdot) : w \in R(l)\right] \geq \Pr\left[b \leftarrow \mathsf{OUT}_{\mathtt{V}}^{\mathtt{P}',\mathtt{V}}((l,y), r|l,\cdot) : b = 1\right] - \mathit{neg}(\eta)$$

A non-interactive proof system [12] is a special type of proof system that consists of a single message from prover to verifier. However, if the entire protocol is under the control of the prover, it is difficult to prohibit it from "proving" false theorems. For this reason, we assume the existence of a *common reference string* of randomness, readable by both prover and verifier but writable by neither. Intuitively, the string provides a random challenge to the prover, which restricts its control over the protocol and keeps it honest:

Definition 27 (Non-interactive proof system) *Let* $f : \mathcal{N} - > \mathcal{N}$ *be a polynomial. Then* $(f, \mathtt{P}, \mathtt{V})$ *is a* non-interactive proof system *for a language* $L \in \mathcal{NP}$ *with witness relation* L_R *if*

1. *Completeness: For all* $l \in L$*, all* $w \in R(l)$*, and all reference strings* $\sigma \in \{0,1\}^{f(|l|)}$*,*

$$\mathtt{V}(l, \mathtt{P}(l, w, \sigma), \sigma) = 1$$

2. *Soundness: For all malicious provers* $\mathtt{P}'$*, then*

$$\begin{aligned} \Pr[\quad & \sigma \leftarrow \{0,1\}^{f(\eta)}; \\ & (x, p) \leftarrow \mathtt{P}'(\sigma) : \\ & x \notin L \ \mathit{but}\ \mathtt{V}(x, p, \sigma) = 1 \quad] \leq \mathit{neg}(\eta) \end{aligned}$$

We can also modify the definition of zero-knowledge for the special setting of non-interaction [11, 60, 12, 14]. Simulation is possible if the common, supposedly-random, reference string is deliberately chosen by the simulator $\mathtt{S}$:

Definition 28 (Non-interactive zero-knowledge (NIZK) proof system) $(f, \mathtt{P}, \mathtt{V}, \mathtt{S})$ *is a* non-interactive zero-knowledge proof system *[12] for a language* $L \in \mathcal{NP}$ *with witness relation* L_R *if*

1. $(f, \mathtt{P}, \mathtt{V})$ *is a non-interactive proof system for* L*, and*

2. *Zero-Knowledge: For all malicious verifiers* $\mathtt{V}'$, $\mathbf{Exp_1}(\eta) \cong \mathbf{Exp}(\eta)$ *where*

$$\begin{aligned}
\mathbf{Exp_1}(\eta) = \Pr[\quad & \sigma \leftarrow \{0,1\}^{f(|\eta|)}; \\
& (l, w) \leftarrow \mathtt{V}'(\sigma); \\
& p \leftarrow \mathtt{P}(l, w, \sigma) : \\
& \mathtt{V}'(p) \qquad], \\
\mathbf{Exp_2}(\eta) = \Pr[\quad & \sigma \leftarrow \mathtt{S}(1^\eta); \\
& (l, w) \leftarrow \mathtt{V}'(\sigma); \\
& p \leftarrow \mathtt{S}'(l) : \\
& \mathtt{V}'(p) \qquad]
\end{aligned}$$

(Both the simulator $\mathtt{S}$ *and the malicious verifier* $\mathtt{V}'$ *keep state between their invocations.)*

Definition 27 above prevents a malicious prover from producing a proof of a false theorem. It does not, however, prevent a malicious prover from modifying a proof of one (true) theorem into a valid-seeming proof of another (false) theorem. In a manner analogous to chosen-ciphertext security, we wish to prevent the corrupt prover from producing a false proof even after having seen proofs for theorems of its choice. For technical reasons, the definition gives the adversary simulated proofs instead of real ones.[3] However, this weaker definition will suffice for our purposes.

Definition 29 (Non-malleable NIZK (NM-NIZK) proof system) $(f, \mathtt{P}, \mathtt{V}, \mathtt{S})$ *is a* non-malleable *non-interactive zero-knowledge proof system [59] for a language* L *if*

1. *There exists a non-interactive proof system* $\pi = (f', \mathtt{P}', \mathtt{V}')$ *for* L*, and*

2. *For all adversaries* $\mathtt{A}$ *and all polynomial-time relations* R'*, there exists an oracle-calling machine* $\mathtt{M}$ *such that* $\mathbf{Exp_1}(\eta) \cong \mathbf{Exp_2}(\eta)$ *where*

$$\begin{aligned}
\mathbf{Exp_1}(\eta) = \Pr[\quad & \Sigma \leftarrow \mathtt{S}(1^\eta); \\
& (l', p', a) \leftarrow \mathtt{A}^{\mathtt{S}(\cdot,\Sigma,\kappa)}(l, p, \Sigma) : \\
& p' \text{ not produced by } \mathtt{S}_2 \wedge (\mathtt{V}(x', p', \Sigma) = 1) \wedge R(x', a) \quad], \\
\mathbf{Exp_2}(\eta) = \Pr[\quad & \sigma \leftarrow \{0,1\}^\eta; \\
& (x', p', a) \leftarrow \mathtt{M}^{\mathtt{A}}(\sigma) : \\
& (\mathtt{V}(x', p', \sigma) = 1) \wedge R(x', a) \qquad]
\end{aligned}$$

Here, the algorithm $\mathtt{M}$ plays the role of the simulator, and produces the new proof p' with access only to $\mathtt{A}$ and not to the source of valid (seeming) proofs.[4]

[3] Real proofs require a witness, which may not be simulatable.

[4] We note that this is stronger than the form of non-malleability originally used to implement chosen-ciphertext security [59]. That form of this definition allowed the adversary access to only one externally-generated valid proof. Although we will be using it in much the same way, we require non-malleability even when the adversary has access to many proofs. Also, an implementation of this stronger form was also given in the original paper [59].

3.3 Our Implementation

In this section, we propose an implementation of a registration-based plaintext-aware encryption scheme. That is, we provide a protocol for sender-key creation and registration, and algorithms for receiver-key generation, encryption, and decryption. The scheme must provide plaintext awareness when the registration authority is honest and chosen-ciphertext security when the registration authority is corrupt.

The intuition of our implementation is simple:

- Following Rackoff and Simon [57], we make the encryption of a message depend on the public keys of both sender and receiver. In particular, we ensure that the ciphertexts of our scheme can be decrypted using the private keys of either receiver *or* sender.
- Our registration process requires that the sender prove to the registration authority (in a zero-knowledge way) knowledge of their private keys. Thus, a successful registration ensures that the registrant knows enough information to decrypt any ciphertext they create. However, because the registration process is zero-knowledge, no registration authority (honest or corrupt) can gain any information about the sender's secret key.
- Lastly, we use digital signatures to ensure that only the registrant can make valid ciphertexts. Thus, valid ciphertexts must come only from parties that can prove possession of knowledge sufficient to decrypt.

The actual encryption mechanism is based on the schemes of Naor and Yung [52] and Sahai [59]. In those schemes, the receiver's public keys is a triple of two public keys for some semantically-secure encryption scheme and a reference string for a NIZK proof system. The sender encrypts the plaintext under both component keys individually. It sends not only the two resulting ciphertexts, but also a non-interactive zero-knowledge proof (relative to the reference string in the receiver's public key) that they do in fact contain the same plaintext. If the NIZK proof system is also non-malleable, then the resulting encryption scheme is chosen-ciphertext secure [59].

Our scheme takes this one step further by adding a self-referential "twist:" the sender now encrypts in both of the receiver's keys and in one of his own. The NIZK proof shows that all three contain the same plaintext. Lastly, we require that the sender sign each ciphertext with some secure signature scheme. Thus, if a ciphertext is well-formed (meaning that the proof and signature are both valid) then the sender must know their own key and be able to decrypt it.

More formally, our scheme $\mathcal{S} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{RU}, \mathcal{RA})$ uses the following four cryptographic primitives as building-blocks:

- $(\mathsf{G}, \mathsf{E}, \mathsf{D})$, a semantically secure cryptosystem in the sense of [25].

- $(f, \mathtt{P}, \mathtt{V}, \mathtt{S})$, a non-malleable NIZK proof system for NP in the sense of [59], where $\mathtt{P}$ is the proving algorithm, $\mathtt{V}$ is the verification algorithm, $\mathtt{S}$ is the simulator, and $f(\eta)$ is the length of the reference string for security parameter η.

- a zero-knowledge proof of knowledge system for NP [8, 24, 26], and

- A secure signature scheme $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$ [27].

Given these, the scheme $\mathcal{S}$ is as follows:

- $\mathcal{G}$ (receiver key generation): Generate $(\mathsf{e}_1, \mathsf{d}_1)$ and $(\mathsf{e}_2, \mathsf{d}_2)$ according to $\mathsf{G}(1^\eta)$. Pick a random σ from $\{0,1\}^{f(\eta)}$. The public (receiver's) key is $\mathsf{e}_r = (\mathsf{e}_1, \mathsf{e}_2, \sigma)$ and the secret key is $\mathsf{d}_r = (\mathsf{e}_r, \mathsf{d}_1, \mathsf{d}_2)$.

- $\mathcal{RU}$ and $\mathcal{RA}$ (sender key-generation and registration): First, $\mathcal{RU}$ generates $(\mathsf{e}_3, \mathsf{d}_3)$ according to $\mathsf{G}(1^\eta)$. Next, generate (s, v) according to G_{sig}. The public (sender's) key is $\mathsf{e}_s = (1^\eta, \mathsf{e}_3, \mathsf{v})$, and the private key is $\mathsf{d}_s = (\mathsf{e}_s, \mathsf{d}_3, \mathsf{s})$. Next, $\mathcal{RU}$ engages RA in a zero-knowledge proof of knowledge for d_3 and s. If the zero-knowledge proof of knowledge terminates correctly, $\mathcal{RA}$ outputs e_s and $\mathcal{RA}$ output $(\mathsf{e}_s, \mathsf{d}_s)$. Otherwise, they both output $\perp$.

  Recall that we assume the key-registration process to be a secure atomic operation. In particular, we assume that the adversary cannot interfere with the protocol executions of honest parties (unless it is the ostensible user or authority). In particular, we assume that the adversary cannot execute the protocol concurrently with other (honest) executions, or modify the messages of a protocol between two honest users.

- $\mathcal{E}$, on input $(m, (\mathsf{e}_1, \mathsf{e}_2, \sigma), ((1^\eta, \mathsf{e}_3, \mathsf{v}), \mathsf{d}_3, \mathsf{s}))$ first computes $\mathsf{c}_1 = \mathsf{E}(\mathsf{e}_1, m)$, $\mathsf{c}_2 = \mathsf{E}(\mathsf{e}_2, m)$, and $\mathsf{c}_3 = \mathsf{E}'(\mathsf{e}_3, m)$. Then, it computes π, a non-malleable NIZK proof against reference string σ that $\mathsf{c}_1, \mathsf{c}_2$, and c_3 all encrypt the same message relative to $\mathsf{e}_1, \mathsf{e}_2$, and e_3, respectively. It then creates $\mathsf{sig} \leftarrow \mathsf{S}_{sig}((\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi), s)$, and outputs $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig})$.

- $\mathcal{D}$, on input $((\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig}), ((\mathsf{e}_1, \mathsf{e}_2, \sigma), \mathsf{d}_1, \mathsf{d}_2), (1^\eta, \mathsf{e}_3, \mathsf{v}))$ first determines if the signature sig is valid for message $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$ with respect to verification key v. If not, it outputs $\perp$. Otherwise, it then determines if π is a valid proof (relative to the reference string σ) that $\mathsf{c}_1, \mathsf{c}_2$ and c_3 are encryptions of the same message under $\mathsf{e}_1, \mathsf{e}_2$ and e_3, respectively. If so, it outputs $\mathsf{D}'(\mathsf{d}_1, \mathsf{c}_1)$. Otherwise, it outputs $\perp$.

3.3.1 Security of $\mathcal{S}$

Theorem 6 *The scheme $\mathcal{S}$ is registration-based plaintext-aware encryption scheme.*

We prove this via several lemmas.

Lemma 7 *$\mathcal{S}$ satisfies registration completeness.*

Proof. By definition, the honest registrant knows the witness d_d for theorem e_d. Hence, the completeness property of the proof-of-knowledge scheme ensures that the registration authority will accept the theorem and proof almost all of the time. ■

Lemma 8 *$\mathcal{S}$ satisfies encryption completeness.*

Proof. If the sender is honest, then it produces $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig})$ where c_1, c_2 and c_3 all contain the same plaintext m and π is an honest proof of that fact. Thus, the signature sig will always verify and the completeness property of the NIZK proof system ensures that the proof π will almost always be accepted. Thus, the decryption algorithm will almost always output $\mathsf{D}(\mathsf{c}_1, \mathsf{d}_r, \mathsf{e}_s)$. Due to the fact that $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is a public-key encryption scheme, this decryption will reveal the intended plaintext. ■

Lemma 9 *$\mathcal{S}$ satisfies honest security.*

Proof. We will prove chosen-ciphertext security by the contrapositive. Suppose there is an adversary $\mathtt{A}$ that succeeds in an adaptive chosen ciphertext attack against an honest sender and an honest recipient. In particular, we will give two algorithms, R and R' respectively, and we will prove that one of the two must break the underlying encryption scheme.

R uses the adversary A to break the semantic security of the encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ as follows:

1. It receives as input $(\mathsf{e}, 1^\eta)$.

2. First, we create the receiver's public key $\mathsf{e}_r = (\mathsf{e}_1, \mathsf{e}_2, \sigma)$ and the sender's public key $\mathsf{e}_s = (1^\eta, \mathsf{e}_3, \mathsf{v})$ as follows. Pick a at random from $\{1, 2\}$. Set e_{3-a} to be e and set $(\mathsf{e}_a, \mathsf{d}_a) \leftarrow \mathsf{G}(1^\eta)$. Generate σ according to the simulator S for the NIZK proof system. Set $(\mathsf{e}_3, \mathsf{d}_3) \leftarrow \mathsf{G}(1^\eta)$. Set $(\mathsf{s}, \mathsf{v}) \leftarrow \mathsf{G}_{sig}(1^\eta)$.

3. Run $\mathtt{A}$ on input $(\mathsf{e}_r, \mathsf{e}_s)$.

   (a) Whenever $\mathtt{A}$ asks for a decryption of $(c_1', c_2', c_3', \pi', \mathsf{sig}')$, encrypted with sending key e', we verify the signature sig' using the verification key in e'. We check the correctness of π' using $\mathtt{V}$. If it verifies, we decrypt c_a' using d_a and output that as the result. Otherwise we return $\perp$.

   (b) Whenever $\mathtt{A}$ asks for the encryption of $(m', (\mathsf{e}_1', \mathsf{e}_2', \sigma'))$, we encrypt m' under e_1', e_2' and e_3 to get c_1', c_2' and c_3' respectively. We create the NIZK proof π' that all three contain the same ciphertext, and sign (c_1', c_2', c_3', π') with s to get sig'. We return $(c_1', c_2', c_3', \pi', \mathsf{sig}')$.

4. Eventually $\mathtt{A}$ will output (m_0, m_1). Output (m_0, m_1) and obtain challenge c. For notation later, let us say that m_β is the message c encrypts.

5. We then simulate the ciphertext challenge for $\mathtt{A}$. Pick b at random from $\{0, 1\}$. Let $\mathsf{c}_a \leftarrow \mathsf{E}'(\mathsf{e}_a, m_b)$, and set $\mathsf{c}_{3-a} \leftarrow c$. With probability $1/2$, let $\mathsf{c}_3 \leftarrow \mathsf{E}'(\mathsf{e}_3, m_b)$ and otherwise, let $\mathsf{c}_3 \leftarrow \mathsf{E}'(\mathsf{e}_3, m_{1-b})$. Fake the NIZK proof π using the simulator $\mathtt{S}$, and sign the ciphertexts and proof with s.

6. Run $\mathtt{A}$ on input $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$.

7. Again, whenever $\mathtt{A}$ asks for a decryption we check the signature, verify the proof, and decrypt using d_a. Whenever $\mathtt{A}$ asks for an encryption, we create one as above.

8. Eventually $\mathtt{A}$ outputs an answer b'. If $b = b'$, output b'. Otherwise, output a random bit.

There are three kinds of input the adversary can get.

I. First, it is possible that $\mathsf{c}_1, \mathsf{c}_2$, and c_3 all encrypt the same message m_β. In this case, the input given to the adversary is indistinguishable from the input in the real attack the adversary succeeds in. Thus, the adversary must return β with probability $1/2 + \epsilon$, where ϵ is some non-negligible function of η.

II. Second, it may be that c_1 and c_2 both encrypt the same message m_β but c_3 encrypts $m_{1-\beta}$. Let x be the probability that the adversary returns β in this case.

III. Finally, it may be that c_1 and c_2 encrypt different messages. Note that there are two subcases:

   - c_a and c_3 encrypt the same message while c_{3-a} encrypts the other, and
   - c_{3-a} and c_3 encrypt the same message while c_a encrypts the other

   These two cases are indistinguishable to the adversary. Since the adversary cannot make any proofs of false theorems, the oracle will return $\perp$ if the adversary ever makes a decryption query when c_1 and c_2 encrypt different messages. Thus, the case $a = 1$ and the case $a = 2$ give the same distribution. (See [59]. This is just like one of the main details from Sahai's proof that his scheme is CCA2-secure.)

   Let $m_{\beta'}$ be the message encrypted in c_3, and let y be the probability that the adversary returns β' in this case.

This reduction is parameterized by the values x and y, both of which can be chosen by the adversary. However, we will show that the only value of interest to us is x. In fact, we will show that for almost all values of x, the above reduction violates the semantic security of $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. However, the reduction R will not work for certain values of x, so we will show that in those cases a different algorithm R' will.

To begin: what is the probability that R returns the correct answer? Again, we consider two cases: when $b = \beta$ and when $b \neq \beta$:

- In the case that $b = \beta$, the adversary sees an input of type I with probability 1/2. When the adversary sees an input of type I, R is correct with probability $\left(\frac{1}{2} + \epsilon\right) + \frac{1}{2}\left(\frac{1}{2} - \epsilon\right) = \frac{3}{4} + \frac{\epsilon}{2}$. If the adversary does not see an input of type I even though $b = \beta$, then it sees an input of type II. In this case, R is correct with probability $x + (1-x)/2$ (since whenever the adversary returns β in an input of type II, R is correct, and the rest of the time, R is correct with probability 1/2). Thus, the total probability that R is correct when $b = \beta$ is $\frac{1}{2}\left((3/4 + \epsilon/2) + (1/2 + x/2)\right)$.

- Now let us examine the case that $b \neq \beta$. Any time this is true, we give input type III to the adversary. However, with probability 1/2, m_b is encrypted in c_3 and with probability 1/2, m_{1-b} is encrypted in c_3. (Recall, these two cases are indistinguishable to the adversary.) Thus, the adversary returns b with probability $\frac{1}{2}y + \frac{1}{2}(1-y)$, and otherwise returns $1-b$. In this case, our total probability of being correct is $(y/4) + (1-y)/4 = 1/4$, since we are only correct when the adversary returns $1-b$, and then, only half the time.

Taking into account all cases, the probability that R is correct is

$$\frac{1}{2}\left(\frac{1}{2}(3/4 + \epsilon/2) + \frac{1}{2}(1/2 + x/2) + \frac{1}{4}\right)$$

This expression evaluates to

$$\frac{3}{16} + \frac{\epsilon}{8} + \frac{1}{8} + \frac{x}{8} + \frac{1}{8} = \frac{7}{16} + \frac{\epsilon + x}{8}.$$

Now if $\epsilon + x$ is non-negligibly different from 1/2 then the above expression is also, and R violates the semantic security of $(\mathsf{G}, \mathsf{E}, \mathsf{D})$. If $x \approx 1/2 - \epsilon$, on the other hand, then we can use $\mathtt{A}$ to break the security of $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ in a different way. Let R' be the algorithm that, on input e, operates as follows:

1. Generate $(\mathsf{e}_1, \mathsf{d}_1)$ and $(\mathsf{e}_2, \mathsf{d}_2)$ by running $\mathsf{G}'(1^\eta)$. Generate σ according to the simulator S for the NIZK proof system. Generate (s, v) by running G_{sig}. Set e_3 to e.

2. Run $\mathtt{A}$ on input $((\mathsf{e}_1, \mathsf{e}_2, \sigma), (\mathsf{e}_3, \mathsf{s}))$.

   (a) Whenever $\mathtt{A}$ asks for a decryption query, verify the proof and signature. If both verify, we decrypt c_1 using d_1 and output that as the result. Otherwise we return $\perp$.

   (b) Whenever $\mathtt{A}'$ makes an encryption query, we encrypt, create the proof, and sign as in the previous reduction.

3. Obtain m_0, m_1 as the output of $\mathtt{A}$. Output (m_0, m_1) and obtain challenge c. For notation later, let us say that m_β is the message c encrypts.

4. Pick b at random from $\{0,1\}$. Let $\mathsf{c}_1 \leftarrow \mathsf{E}'(\mathsf{e}_1, m_b)$, let $\mathsf{c}_2 \leftarrow \mathsf{E}'(\mathsf{e}_2, m_b)$, and let c_3 be c.

5. Fake the NIZK proof π using the simulator $\mathtt{S}$.

6. Sign with s to make signature sig.

7. Run $\mathtt{A}$ on input $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig})$. Again, whenever $\mathtt{A}$ asks for a decryption, we check the signature and proof, and decrypt using d_1. Whenever $\mathtt{A}$ asks for an encryption, we create one as above. Eventually $\mathtt{A}$ outputs an answer b'. Output b'.

The proof that R' works is simple. If R' picks $b = \beta$ then A outputs b with probability $1/2 + \epsilon$. If R' picks $b \neq \beta$ then A sees input type II and so it outputs b with probability $x = 1/2 - \epsilon + \nu'$ where ν' is (positively or negatively) negligible. Thus in either case, we output β with probability at least $1/2 + \epsilon - |\nu'/2|$. ■

Lemma 10 *$\mathcal{S}$ enjoys plaintext awareness.*

Proof. There are two cases. In this first case, suppose that $\mathtt{X} = \mathsf{RU}$. We need an extractor $\mathtt{E}_{\mathsf{RU}}$ such that $\mathtt{E}_{\mathsf{RU}}$ will accurately predict the output of the receiver's decryption algorithm. Let $\mathtt{E}_{\mathsf{RU}}$ be the algorithm that simply returns $\perp$. If this $\mathtt{E}_{\mathsf{RU}}$ is not correct a non-negligible proportion of the time, then the adversary has a non-negligible probability of forging a signature. Because the digital signature scheme is secure against the chosen-plaintext attack, this is a contradiction.

Let us show this formally. Suppose that for some adversary, the extractor $\mathtt{E}_{\mathsf{RU}}$ is incorrect with a polynomial probability. Then with that same polynomial probability, the adversary produces something that does not decrypt to $\perp$:

$$\begin{aligned}
&\exists \text{ adversary } \mathtt{A}, \exists \text{ PPT ally } \mathtt{L}, \exists \text{ polynomial } q, \text{for infinitely many } \eta \\
&\Pr[\quad (\mathsf{e}_r, \mathsf{d}_r) \leftarrow \mathsf{G}(1^\eta); \\
&\qquad \mathsf{e}_{\mathtt{X}} \leftarrow \mathsf{OUT}_{\mathsf{RA}}^{\mathtt{X},\mathsf{RA}}(\mathsf{e}_r, \cdot | 1^\eta, \cdot)\,; \\
&\qquad h \xleftarrow{H} \mathtt{A}; \\
&\qquad c \leftarrow \mathtt{A}^{\mathtt{L}'_{\mathsf{d}_{\mathsf{RU}}}(\cdot), \mathsf{D}(\cdot, \mathsf{d}_r, \cdot)}(\mathsf{e}_{\mathtt{X}}, \mathsf{e}_r) : \\
&\qquad \mathsf{D}(c, \mathsf{d}_r, \mathsf{e}_{\mathtt{X}}) \neq \perp \text{ given that } c \notin \mathtt{L}_c) \quad] \geq \tfrac{1}{q(\eta)}
\end{aligned}$$

We use $\mathtt{A}$ and $\mathtt{L}$ to construct an adversary $\mathtt{A}'$ that will violate the security of the signature scheme $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$. This new adversary $\mathtt{A}'$ works as follows:

- On input 1^η and v, $\mathtt{A}'$ will pick $(\mathsf{e}_1, \mathsf{d}_1) \leftarrow \mathsf{G}'(1^\eta)$, $(\mathsf{e}_2, \mathsf{d}_2) \leftarrow \mathsf{G}'(1^\eta)$, $(\mathsf{e}_3, \mathsf{d}_3) \leftarrow \mathsf{G}'(1^\eta)$, $\sigma \leftarrow \{0,1\}^{f(\eta)}$. It gives $\mathsf{e}_{\mathsf{RU}} = (\mathsf{e}_3, \mathsf{v})$ and $\mathsf{e}_r = (\mathsf{e}_1, \mathsf{e}_2, \sigma)$ to $\mathtt{A}$.

- If $\mathtt{A}$ makes a call to $\mathsf{D}(\cdot, \mathsf{d}_r, \cdot)$ with c' and e'_s, $\mathtt{A}'$ will decrypt as usual. That is, it checks the signature component in c' with the verification key in e' and the proof component of c' against the reference string in d_r. It then decrypts with d_1 and returns the result.

- If $\mathtt{A}$ makes a call to $\mathtt{L}'_s(\cdot)$, then $\mathtt{A}'$ simulates $\mathtt{L}$ on the history of $\mathtt{A}$ (which it is also simulating). It then takes the resulting (m, e'_r) and creates the return value c' as thus:
  - It encrypts m in e_3 from e_s. It also encrypts m in the e'_1 and e'_2 from e'_r.
  - It proves, relative to the reference string σ' from e'_s, that all three of these ciphertexts contain the same plaintext.
  - It then sends the three ciphertexts and the proof to the signing oracle $\mathsf{S}_{sig}(\mathsf{s}, \cdot)$ to get a signature.

  The three ciphertexts, proof, and signature from the oracle are jointly returned to $\mathtt{A}'$.

- When $\mathtt{A}$ returns a ciphertext $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig})$, return $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$ as the message m and sig as the signature.

Suppose that $\mathsf{D}((\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig}), \mathsf{d}_r = ((\mathsf{e}_1, \mathsf{e}_2, \sigma), \mathsf{d}_1, \mathsf{d}_2), \mathsf{e}_{\mathsf{RU}} = (\mathsf{e}_3, \mathsf{v}))$ returns anything other than $\perp$. Then it must be that $\mathsf{V}_{sig}((\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi), \mathsf{sig}, \mathsf{v}) = 1$. Furthermore, we know that $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig})$ cannot be a ciphertext returned by the ally. Thus, either $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$ is not a message sent to the signature oracle, or it was but sig is not the signature that it returned for that message. However, we know that every message has a unique signature. Hence, it must be that $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$ was not signed by the signing oracle, and the adversary has created a new message $(\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi)$ and a valid signature sig for it. Thus, if $\mathtt{E}_{\mathsf{RU}}$ does not predict the receiver's output (except for a negligible fraction of the time) then $\mathtt{A}'$ violates the security of the digital signature scheme.

In the other case, $\mathtt{X} = \mathtt{A}$, and the adversary registered $\mathsf{e}_{\mathtt{A}}$. Then the simulator $\mathtt{E}_{\mathtt{A}}$ is as follows:

- On input $(h, (\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3, \pi, \mathsf{sig}), \mathsf{e}_r = (\mathsf{e}_1, \mathsf{e}_2, \sigma), \mathsf{e}_{\mathtt{A}} = (\mathsf{e}_3, \mathsf{v}))$, we use h to rewind the adversary to the point where $\mathtt{A}$ engages in key registration with RA. This key-registration process is the execution of a zero-knowledge proof of knowledge for the private key. Hence, there exists an extractor $\mathtt{E}$ for the proof system that will be able to extract the public keys given oracle access to the (possibly malicious) prover $\mathtt{A}$. By assumption, $\mathtt{A}$ is able to register its public key, so we run the extractor (with oracle access to $\mathtt{A}$, rewound to the registration step) to extract $\mathsf{d} = (\mathsf{d}_3, \mathsf{v})$, the secret key associated with $\mathsf{e}_{\mathtt{A}}$.

- We then verify the signature sig and verify the proof π; if either are invalid, we output $\perp$.

- Otherwise, we use d_3 to decrypt c_3 and give the result as the answer.

From the extractibility property of the proof system, d must be a secret key relative to e. However, we need to show that the decryption under d will always be the same as the decryption under d_r. If the signature sig or the proof π in c fails to verify, then certainly we are correct to output $\perp$. If both verify, however, we need to show that (except with negligible probability) that $\mathsf{c}_1, \mathsf{c}_2$, and c_3 all encrypt the same message. However, this is the same as showing that the proof system is non-malleable even when the adversary has access to a polynomial number of valid proofs, which is our assumption. Hence, if the proof π verifies then all three ciphertexts will contain the same plaintext. By decrypting c_3 with d_3 we will produce the same plaintext as the decryption algorithm would by decrypting c_1 with d_1. ■

Chapter 4

Diffie-Hellman Key Exchange in the Dolev-Yao model

In this chapter, we introduce consider the extension of the Dolev-Yao model to include the Diffie-Hellman key-agreement scheme. This widely-used scheme relies upon properties of a commutative algebra and, as opposed to the basic Dolev-Yao algebra, requires non-freeness of terms.

We introduce an extension of the Dolev-Yao algebra that allows protocol designers to express the Diffie-Hellman scheme as part of their protocols. This extension allows the adversary to perform any efficient computation on Diffie-Hellman values, and considers the non-freeness that they (and the Diffie-Hellman scheme itself) introduce. We also formulate a formal version of the Diffie-Hellman assumption in the extended Dolev-Yao model that is targeted for the analysis of real-world protocols.

Lastly, we consider the computational soundness of our extension. In particular, we introduce a computational interpretation of attacks in the Dolev-Yao model. We show that under this interpretation, any Dolev-Yao attack that violates the formal version of the Diffie-Hellman assumption maps to a computational algorithm that violates the computational Diffie-Hellman assumption also.

4.1 Overview

In the two previous sections, we have shown that it is possible to ground a large part of the standard Dolev-Yao model in the more concrete world of computational cryptography. Many popular real-world protocols will not be affected by this grounding, however, because they do not yet fit into the Dolev-Yao model.

Consider the Transport Layer Security (TLS) [19] protocol. This widely-used key-exchange protocol does not use public-key encryption at all, but requires symmetric encryption, hashing, digital signatures, and the Diffie-Hellman key-exchange scheme. At its most distilled form, this protocol is a series of four messages between a client (C) and server (S)[1]:

1. $C \longrightarrow S : C$
2. $S \longrightarrow C : S\ [g^x]_{K_S}$
3. $C \longrightarrow S : [g^y]_{K_C}\ \{\!|T_1\ C\ S|\!\}_{K'}$
4. $S \longrightarrow C : \{\!|T_2\ C\ S|\!\}_{K'}$

where

- T_1, T_2 are fixed tags to distinguish the third message from the fourth,
- $[M]_{K_X}$ is the message M together with a signature that can be verified using the verification key K_X,
- $\{\!|M|\!\}_{K'}$ is the message M encrypted with the symmetric key K',
- g is a generator for some group G,
- x, y are randomly chosen elements of $\{1, 2, \ldots |G|\}$, and
- K' is a symmetric key created by hashing the value g^{xy}.

The first three of these operations—symmetric encryption, digital signatures, and hashing—are not totally foreign to the Dolev-Yao model. Like public-key encryption, they are general definitions that can be satisfied by a variety of implementations. It is reasonable to treat them as public-key encryption was treated: represented by formal operations which allow the formal adversary only limited abilities while also showing that the same limitation applies to the computational adversary.

We anticipate that there will soon appear an analysis, like that of Chapter 2, demonstrating computational soundness for these three operations, and hence do not focus on them here. (We will require a strong security condition on hashing later in this chapter, however.) Computational soundness for the Diffie-Hellman scheme, on the other hand, seems to be tricker:

- Firstly, it is not obvious which is the best way to even represent the Diffie-Hellman scheme in the Dolev-Yao model. The Dolev-Yao model assumes that the algebra is free, and one can conceive of encoding operations that preserve this freeness. The Diffie-Hellman scheme, on the other hand, *requires* non-freeness: the correct operation of the scheme requires that $(g^x)^y = (g^y)^x$. Given this, it is not obvious how to capture the non-free group and its operations in the (traditionally) free Dolev-Yao algebra.[2]

[1]The full TLS protocol is much more elaborate, primarily to ensure robustness and compatibility. It actually has two basic forms, one based on Diffie-Hellman and one based on public-key encryption. We show here only the security-relevant messages of the Diffie-Hellman version.

[2]There are approaches to Diffie-Hellman in formal cryptography which do not use the Dolev-Yao model and which use non-free cryptosystems. See Lynch[40] for an example.

- Also, it is not clear what powers the Dolev-Yao adversary has with respect to the Diffie-Hellman scheme. Some candidate abilities spring quickly to mind:
  - Just as it can always create fresh nonces and keys, the Dolev-Yao adversary should be able to create fresh Diffie-Hellman values.
  - Since the Diffie-Hellman scheme requires the honest participants to calculate the group operation (and exponentiation) the adversary must be able to do so as well.

  As reasonable as these two operations are, it is unreasonable to assume that an adversary would be limited to them—in either model.

There are many possible resolutions to these difficulties, and to choose among them one must decide whether to use the Dolev-Yao model to find flaws or to generate proofs. If one wishes to find flaws (but does not necessarily need completeness of the flaw-finding algorithm) then one can simply select any convenient representation and any convenient set of Dolev-Yao operations. If these are chosen carefully, then automated flaw-finding algorithms can be applied to uncover any flaws from a given class. (This is the approach taken in, for example, [56].)

We, however, take a different tactic in this chapter. We wish to use the Dolev-Yao model to generate high-level proofs that remain valid in the computational model. To do this, we will take an approach much like that in Chapter 2: we present a syntactic limitation on the Dolev-Yao adversary, and prove that it corresponds to an analogous restriction on the computational adversary. However, we will now do this in the absence of any obvious closure operation. To define a new closure operation for Diffie-Hellman would require that the adversary's powers be rigorously enumerated, and as mentioned above, we do not know what this enumeration should be.

However the closure operation in Chapters 1 and 2 was only an intermediate definition between the assumptions of the Dolev-Yao model and the computational definition of chosen-ciphertext security. Such an intermediate step is not strictly necessary. Our intent in the present chapter is to disregard this intermediate step and connect the Dolev-Yao model directly to the Diffie-Hellman assumption. With this decided, our resolution to the two difficulties above becomes clear.

The Diffie-Hellman assumptions stipulate that given computation is beyond the reach of *any* reasonable adversary, and so there is no need to decide on explicit powers for the Dolev-Yao adversary. If we represent the Diffie-Hellman assumption properly, then it will remain sound even if we allow the Dolev-Yao adversary to perform efficient computations. This is exactly what we will do: rather than limit the adversary to some artificial set, we allow it to perform any efficient calculation.[3]

We develop our approach in several parts. First, we expand the Dolev-Yao model to incorporate the Diffie-Hellman key-exchange scheme. This expansion itself takes multiple steps:

[3]As an added benefit, this ensures that our results will remain valid even if the adversary is limited in order to aid the discovery of flaws.

- First, the algebra $\mathcal{A}$ is itself expanded. We add operations for symmetric encryption, signatures, and hashing. Because we later allow the adversary to perform arbitrary efficient computations, we will also include a new symbol to represent each such calculation. However, we insist that the Dolev-Yao algebra remain free. As mentioned above, we cannot continue to assume non-freeness when we add arbitrary computation. Furthermore, the Diffie-Hellman scheme requires non-freeness to even operate. However, both of these statements refer to non-freeness in the *computational* setting. Thus, we can continue to assume freeness in the Dolev-Yao setting and non-freeness in the computational setting. We reconcile the freeness or lack thereof of the two setting when we show how attacks in the formal world map to attacks (or algorithms) in the computational one.

- We perform this reconciliation when we revise the notion of a valid trace. Our new definition serves two purposes: first, it makes explicit the internal operations of the adversary (which Assumption 3 hid behind the closure operation). Second, it captures the realities of the computational non-freeness. That is, a valid trace can now use two different Dolev-Yao terms interchangeably, so long as they have the same computational interpretation.

  We note that as a result valid Dolev-Yao traces are now defined (in part) in terms of the underlying bit-string representation—a departure from the encoding algorithm of Section 1.3.

Given the expanded Dolev-Yao model, it is almost trivial to define the Dolev-Yao representation of the Diffie-Hellman assumption. The Dolev-Yao model, after all, is an alternative model of executions. There is a natural mapping from Dolev-Yao traces to computational algorithms. Given this, there is a similarly natural way to characterize traces that "solve" the Diffie-Hellman problem and to give a syntactic security condition (Property $\mathcal{DH}$, definition 40) that prohibits them. We show this, and that traces that violate condition $\mathcal{DH}$ map to algorithms that solve the computational Diffie-Hellman problem—no matter which cryptographic algorithms are chosen to implement the mapping.

This, however, is not enough. Dolev-Yao protocols are not *a priori* required to be efficiently computable. Hence, traces over infeasible protocols may not map to efficient algorithms. Such algorithms that solve the Diffie-Hellman problem do not necessarily violate the Diffie-Hellman assumption. It is legitimate for traces to violate condition $\mathcal{DH}$ if the honest participants inadvertently aid the adversary in doing so.

We resolve this difficulty by imposing efficiency upon the protocols. In particular, we define what it means for a protocol to be "silent" (Definition 41), a natural and purely syntactic condition on protocols. A silent protocol is one where honest participants will never use a secret Diffie-Hellman value as a message or a plaintext, but only as key material. One the one hand, this condition is natural enough to capture a large class of real-world protocols such as TLS and SSH. On the other hand, this condition allows us to use security properties of the computational algorithms to restrict the honest participants. In particular, we show that traces of silent traces map to efficient algorithms if hashing can be implemented with our friend, the random oracle.

Thus, a trace that violates security condition $\mathcal{DH}$ maps to an algorithm that solves Diffie-Hellman. Furthermore, a trace over a silent protocol can be mapped, via the random oracle, to an efficient algorithm. Hence, the main result of this work that under the random oracle and computational Diffie-Hellman assumptions, there can be no trace over a silent protocol that also violates condition $\mathcal{DH}$. Thus, it is computationally sound to use and assume property $\mathcal{DH}$. (See [32] for an example.)

The use of the random oracle here is analogous to its use in Theorem 1. That is, it is used to show that a connection between the Dolev-Yao and computational models can be made. Just as Chapters 2 and 3 remove the random oracle in the case of the standard Dolev-Yao model, we expect future work to remove the random oracle from this chapter as well. In fact, we will mention some possible approaches to this in Chapter 5.

The rest of this chapter is as follows. First, we quickly review the Diffie-Hellman assumption given in Chapter 1.4, and present a weaker form that will be useful for our present purposes (Section 4.2). We then extend the Dolev-Yao model to include the Diffie-Hellman scheme by first expanding the algebra (Section 4.3.1) and then by expanding the notion of valid traces (Section 4.3.2).

We then define the security condition $\mathcal{DH}$ and the notion of silent protocols. We first "derive" these definitions informally (Section 4.4). We then give the mapping from valid Dolev-Yao traces to algorithms (Section 4.5) and the first of our primary results: a proof that a trace that violates condition $\mathcal{DH}$ maps to an algorithm that solves Diffie-Hellman (Theorem 12). We then restrict our attention to silent protocols (Section 4.6) and show that the random oracle allows us to efficiently simulate their computational interpretations. Thus, traces over silent protocols that violate condition $\mathcal{DH}$ map to algorithms that solve Diffie-Hellman efficiently (Theorem 11).

4.2 The Diffie-Hellman Problem

In this section, we review the Diffie-Hellman key-agreement scheme that was introduced in Section 1.4. All participants of this scheme must agree on a family $\{g_\eta, G_\eta\}_\eta$ (where each G_η is a cyclic group generated by g_η) which is fixed and indexed by the security parameter. When the security parameter η is clear from context, we will refer to G_η and g_η as simply G and g. It is assumed that every element of G_η can be represented with a number of bits polynomial in η.

Two entities A and B can use this family to agree on a secret random value in the following way:

- A value of the security parameter η is chosen via some external mechanism.[4]

[4]In practice, this is either fixed in a standards document or negotiated via a preliminary round of communication.

- A chooses a random element $x \in \{1 \ldots |G_\eta|\}$ and sends to B the value g^x, and
- B chooses a random element $y \in \{1 \ldots |G_\eta|\}$ and sends to A the value g^y.

The random value upon which they have agreed is g^{xy}, which both can calculate:

- A can calculate g^{xy} from x and g^y via $(g^y)^x = g^{xy}$ and
- B can calculate g^{xy} from y and g^x via $(g^x)^y = g^{xy}$.

We will call the values g^x and g^y the *base* Diffie-Hellman values, and call g^{xy} the Diffie-Hellman *secret*.

Note that the scheme provides no authentication. Although A can be sure that the secret value g^{xy} is known only to A herself and the entity that generated y, A cannot tell who that entity is. Authentication and identification must be ensured by some other mechanism.

The scheme is, however, assumed to provide secrecy in the sense that no agent other than A and B should be able to learn the value g^{xy}. This notion can be formalized in two different ways. The stronger formalization is the *decisional* Diffie-Hellman assumption, which is given in Section 1.4. Briefly, this assumption states that no efficient adversary can distinguish the actual Diffie-Hellman key from a random group element:

Definition 30 *The* decisional Diffie-Hellman assumption *for a group family* $\{g_\eta, G_\eta\}_\eta$ *is:*

$$\forall \textit{ PPT algorithms } \mathtt{A}, |\mathbf{Exp}_1(\eta) - \mathbf{Exp}_2(\eta)| \leq \textit{neg}(\eta)$$

where

$$\begin{aligned}
\mathbf{Exp}_1(\eta) = \Pr[\quad & x, y \leftarrow \{1, 2, \ldots, |G_\eta|\}\,; \\
& b \leftarrow \mathtt{A}(1^\eta, G_\eta, g_\eta, g_\eta^x, g_\eta^y, g_\eta^{xy}) : \\
& b = 1 \qquad\qquad\qquad\qquad] \\
\mathbf{Exp}_2(\eta) = \Pr[\quad & x, y, z \leftarrow \{1, 2, \ldots, |G_\eta|\}\,; \\
& b \leftarrow \mathtt{A}(1^\eta, G_\eta, g_\eta, g_\eta^x, g_\eta^y, g_\eta^z) : \\
& b = 1 \qquad\qquad\qquad\qquad]
\end{aligned}$$

This is the form we will assume for this chapter. However, we will also use the original weaker form known as the *computational* Diffie-Hellman assumption, which simply requires that the adversary be unable to produce g^{xy} from only g^x and g^y:

Definition 31 *The* computational Diffie-Hellman problem *over for a group family* $\{G_\eta\}_\eta$ *to produce* $g^x y$ *from input* g^x *and* g^y*. The* computational Diffie-Hellman assumption *is that the computational*

Diffie-Hellman problem is hard if x and y are chosen randomly:

$$\begin{array}{ll} \multicolumn{2}{l}{\forall \textit{ PPT algorithms } \mathtt{A}:} \\ \Pr[& x, y \leftarrow \{1, 2, \ldots, |G_\eta|\}\,; \\ & h \leftarrow \mathtt{A}(1^\eta, G_\eta, g_\eta, g_\eta^x, g_\eta^y) : \\ & h = g_\eta^{xy} \qquad\qquad\qquad] \leq neg(\eta) \end{array}$$

Note that the decisional form of the assumption implies the computational one, and thus is the one usually assumed. We, however, will use the computational Diffie-Hellman assumption in this chapter. To do so, we first need to extend the Dolev-Yao algebra to the point that the assumption can be even expressed.

4.3 Extending the Dolev-Yao Model

In incorporate Diffie-Hellman into the Dolev-Yao model, we expand the Dolev-Yao model in two ways: we enlarge the algebra, and we reconsider the joint definition of the adversary and trace validity.

4.3.1 Extending the Algebra

We will extend and modify the algebra of Chapter 1 in three ways.

We add operators for symmetric encryption, signatures and hashing. This is for two reasons:

1. We will later use properties of the hash algorithm in an essential way.
2. One of our ultimate goals is to enable the analysis of popular protocols such as TLS and SSH, and these protocols use the above operations.

Signing a term is not assumed to hide the message in any fashion. Hashing a term is assumed to result in a key appropriate for symmetric encryption. We will assume that keys for asymmetric encryption, symmetric encryption and signatures are mutually disjoint, and that symmetric encryption keys created through hashing are disjoint from those created directly.

Next, we add an additional type $\mathcal{D}$ for Diffie-Hellman messages (i.e. group elements). We will use d_1, $d_2 \ldots$ as elements of $\mathcal{D}$, and we assume there exists an operation $DH : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ to represent the Diffie-Hellman operation. We denote the range of DH by $\mathcal{D}_{DH}$. In the literature and in practice, the notation g^x is often used as both a computational and formal variable and g^{xy} used instead of the admittedly cumbersome $DH(d_1, d_2)$. In contexts where the model is clear, this overloading of notation presents no difficulty. In this work, however, we are interested in the exact relationship between the formal and computational models, and will use notation to distinguish the two. Thus, we will use the notation d_1, d_2 and $DH(d_1, d_2)$ only for the formal model, and the notation g^x, g^y and g^{xy} only for the computational one.

Lastly, we will want (in the next section) to capture the possible behaviors of the adversary. As mentioned above, we will allow the adversary to perform any efficient computation, which requires us to define what we mean by this.

Definition 32 *A function $f : \mathcal{N} \to \mathcal{R}$ is* noticeable *if there exists a polynomial q such that $f(\eta) \geq \frac{1}{q(\eta)}$ for all sufficiently large η.*

For our purposes, computation can be performed efficiently if there is a machine with a noticeable probability of performing it.

Definition 33 *A function $f : \{0,1\}^* \to \{0,1\}^*$ is* efficient *or* efficiently computable *if there exists a probabilistic Turing machine $\mathtt{M}_f$ so that $\Pr[\mathtt{M}_f(x) = f(x)]$ is noticeable in $|x|$.*

Since we will allow the adversary to perform any efficient computation, we will need a notation to represent output so produced. Therefore, we add to the Dolev-Yao algebra a constructor for each efficiently computable function. However, we will assume that the adversary can only perform these computations on Diffie-Hellman values and only to produce Diffie-Hellman values. (We leave consideration of the adversary's ability to produce values of other types to analyses like that in Chapter 2.)

To combine and formalize these modifications:

Definition 34 *The set of terms $\mathcal{A}$ is (now) assumed to be freely generated from four disjoint sets:*

- *$\mathcal{T} \subseteq \mathcal{A}$, which contains predictable texts,*
- *$\mathcal{R} \subseteq \mathcal{A}$, which contains unpredictable random values,*
- *$\mathcal{K} \subseteq \mathcal{A}$, which contains keys, and*
- *$\mathcal{D} \subseteq \mathcal{A}$, which contains Diffie-Hellman values.*

The set of keys ($\mathcal{K}$) is divided into five disjoint sets:

- *encryption keys ($\mathcal{K}_{Pub}$),*
- *decryption keys ($\mathcal{K}_{Priv}$)*
- *signature keys ($\mathcal{K}_{Sig}$),*
- *verification keys ($\mathcal{K}_{Ver}$), and*
- *keys for symmetric encryption ($\mathcal{K}_{Sym}$).*

We assume a mapping inv which maps one member of a key pair to the other, and a symmetric key to itself. Compound terms are built by the operations:

- *hash* $: \mathcal{A} \to \mathcal{K}_{Sym}$*, representing hashing into keys. We will denote the range of hash by* $\mathcal{K}_{hash}$*.*
- *encrypt* $: (\mathcal{K}_{Sym} \cup \mathcal{K}_{Pub}) \times \mathcal{A} \to \mathcal{A}$*, which represents encryption.*
- *sig* $: \mathcal{K}_{Sig} \times \mathcal{A} \to \mathcal{A}$*, which represents signing a message.*
- *pair* $: \mathcal{A} \times \mathcal{A} \to \mathcal{A}$*, which represents concatenation of terms.*
- $DH : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$*, which represents the Diffie-Hellman operation. (As mentioned previously, we denote the range of DH by* $\mathcal{D}_{DH}$*.)*
- *The operator* $F_f : \mathcal{D}^* \to \mathcal{D}$ *for each efficiently-computable* f*, where* $\mathcal{D}$ *is the set of finite sequences from* $\mathcal{D}$*. Note that although* f *is a function from bit-strings to bit-strings, we are here using it as a function from* $\mathcal{D}$*-sequences to* $\mathcal{D}$*.*

We will write $sig(K, M)$ as $[M]_K$. We also note that a previous definition of "ingredient" still applies:

Definition 15 *If* M*,* N *are two elements of* $\mathcal{A}$*, then* M *is an* ingredient *of* M'*, written* $M \sqsubseteq N$*, if the parse tree of* M *is a sub-tree of the parse tree of* N*.*

That is, the ingredients of a message M are those which must have been used in its creation. In particular, this implies that the values d_1 and d_2 are ingredients of $DH(d_1, d_2)$, as well as ingredients of $F_f(d_1, d_2)$. Also, M is an ingredient of $hash\,(M)$.

We will also need a slight variation on this intuition [63] to denote those messages which could possibly be learned from M:

Definition 35 *We say that* M *is a* subterm *of* N*, written* $M \prec N$*, if:*

- $M = N$*, or*
- *if* $N = N' \, N''$*, then* $M \prec N'$ *or* $M \prec N''$*,*
- *if* $N = \{\!|N'|\!\}_K$*, then* $M \prec N'$*,*
- *if* $N = [N']_K$*, then* $M \prec N'$*,*
- *if* $N = DH(d_1, d_2)$*, then* $M \prec d_1$ *or* $M \prec d_2$*,*
- *if* $N = F_F(d_1, d_2, \ldots, d_n)$*, then there exists a* d_i *so that* $M \prec d_i$*.*

In particular, it is conceivable that one can learn g^x from g^{xy} (if, for example, y is chosen to be 1). Hence, d_1 can possibly be learned from $DH(d_1, d_2)$ and $d_1 \prec DH(d_1, d_2)$. Likewise, the function f might be invertible, and so $d_i \prec F_F(d_1, d_2, \dots d_n)$ for each d_i. Note, however, that K is not a subterm of $\{\!|M|\!\}_K$ or $[M]_K$ (unless K is also a subterm of M). This is because it is impossible to learn a key from a symmetric encryption. Further, all public keys are already known and so it is meaningless to speak of "learning" them from an encryption or signature.

4.3.2 Traces and the Adversary

In this chapter, as in Chapter 1, the adversary and the set of valid traces are defined in terms of each other. One can think of a valid trace as one in which the adversary performs only the allowed actions, or an adversary as an entity that can non-deterministically choose between valid traces. In this chapter, however, we will not use any closure operation to compactly represent the adversary's powers. Hence, the connection between the adversary and trace will be a little more direct.

We first list the powers which we assume the adversary possesses. The adversary can:

- make any predictable text,
- say any key it knows, whether in $\mathcal{K}_{Adv}$, $\mathcal{K}_{Pub}$ or $\mathcal{K}_{Ver}$,
- encrypt any message it knows with any encryption or symmetric key that it knows
- decrypt an encryption given that it knows the decryption key,
- sign any message it knows with any signature key it knows,
- extract the "plaintext" from any signature,
- hash any value it knows,
- make fresh random values, which we represent by allowing it to produce whatever it wants from a distinguished set $\mathcal{R}_{Adv} \subseteq \mathcal{R}$,
- generate new Diffie-Hellman values, which we represent by distinguishing the set $\mathcal{D}_P \subseteq \mathcal{D}$ and allowing the adversary to produce any value in that set. (We assume that $\mathcal{D}_P$ and $\mathcal{D}_{DH}$ are disjoint.)
- perform the Diffie-Hellman operation, given that it knows one of the appropriate exponents. That is, if it knows d_1 and created d_2 (that is, $d_2 \in \mathcal{D}_P$) it can generate $DH(d_1, d_2)$.
- for every efficiently computable function f, we allow the adversary to generate $F_f(d_1, d_2, \dots d_n)$ if it knows d_1, d_2, $\dots d_n$.

We note that although the Dolev-Yao algebra is free, two terms (such as $F_f(d_1, d_2, \ldots d_n)$ and $F_{f'}(d'_1, d'_2, \ldots d'_n))$ may represent the same "real" value. We will need to handle this possibility when we define valid traces, and so need to formalize this notion. This, in turn, requires us to extend the encoding operation to Diffie-Hellman additions to the Dolev-Yao algebra

Definition 36 (Extended encoding) *Let $\eta \in \mathcal{N}$ be the security parameter. Let $t \in \{0,1\}^\omega$ be a random tape, partitioned into a length-η segment for each element of $\mathcal{R}$, $\mathcal{K}$ and $\mathcal{D}$. Let $\{g_\eta, G_\eta\}_\eta$ be a family of cyclic groups. Then for any $M \in \mathcal{D}$, M of the form $DH(d_1, d_2)$, or M of the form $M = F_f(d_1, d_2, \ldots d_n)$, the* encoding *of M, written $[\![M]\!]_\eta^t$, is defined recursively:*

- *If $M \in \mathcal{D}$, then $[\![M]\!]_\eta^t = \langle g^x$, "DH value"$\rangle$ where g is the generator for G_η and the randomness of σ_M is used to generate an x uniformly from $\{1, 2, \ldots |G|\}$. (Note: since each element of G is assumed to have a η-bit representation, there is more than enough randomness to in σ_M to generate the required uniform distribution.) σ_M is being interpreted as the binary representation of a natural number.)*

- *If $M = DH(d_1, d_2)$ then $[\![d_1]\!]_\eta^t$ is $\langle g^x$, "DH value$\rangle$ for some $g^x \in G_\eta$ and $[\![d_2]\!]_\eta^t$ is $\langle g^y$, "DH value$\rangle$ for some $g^y \in G_\eta$. Then $[\![DH(d_1, d_2)]\!]_\eta^t = \langle g^{xy}$, "dh value"$\rangle$. We note that calculating g^{xy} from g^x and g^y may not be efficiently computable, but delay discussion of this issue until Section 4.6.*

- *If $M = F_f(d_1, d_2, \ldots d_n)$, then $[\![M]\!]_\eta^t$ is the mapping from distributions to distributions given by $\left\langle f([\![d_1]\!]_\eta^t, [\![d_2]\!]_\eta^t, \ldots [\![d_n]\!]_\eta^t), \text{"DH value"} \right\rangle$.*

Note that the encoding $[\![F_f(d_1, d_2, \ldots d_n)]\!]_\eta^t$ is given in terms of f and not $\mathtt{M}_f$. Hence, the distribution of $[\![F_f(d_1, d_2, \ldots d_n)]\!]_\eta^t$ will be a fixed value if the same is true for each $[\![d_i]\!]_\eta^t$. This differs from $[\![\{\!|M|\!\}_K]\!]_\eta^t$ which may be a non-trivial distribution for all values of $[\![M]\!]_\eta^t$ and $[\![K]\!]_\eta^t$ are. This is because the encrypting algorithm may be probabilistic, and hence there may be many valid ciphertexts for any given plaintext/key pairs. Therefore, there are many valid encodings of $\{\!|M|\!\}_K$, even for M and K that have unique representations. However, f is defined to be a function, and so the value $f([\![d_1]\!]_\eta^t, [\![d_2]\!]_\eta^t, \ldots [\![d_n]\!]_\eta^t)$ will only be as probabilistic as the individual $[\![d_i]\!]_\eta^t$.

Now that we know what Dolev-Yao values represent, we can formalize their proper usage. In previous chapters, a Dolev-Yao trace was an alternating sequence of adversary queries and the responses of honest parties. A *valid* trace was one in which each adversary query was derivable from the trace's previous messages of via a finite number of atomic operations—represented by the application of a single closure operation. Although this representation is compact, it hides the internal operation of the adversary. This internal reasoning of the adversary will be convenient for our purposes, and so we will use an alternate, equivalent, representation that makes this reasoning explicit. In this alternate representation, each adversary query must be derivable from previous messages via a *single* atomic operation, but the adversary is allowed to apply any finite number of them between participant responses.

We require that the definition of valid traces also handle the inherent non-freeness of the adversary's arbitrary computations. In particular, we will allow the adversary to apply the function f to values d_1, d_2, $\ldots d_n$ but produce a Dolev-Yao message M other than $F_f(d_1, d_2, \ldots d_n)$. We will require, however, that the Dolev-Yao message M represent merely another way of performing the same calculation.

Definition 37 *A* valid explicit Dolev-Yao trace *for a group family $\{q_\eta, G_\eta\}_\eta$ is an alternating sequence:*

$$R_0 \quad Q_1 \quad R_1 \quad Q_2 \quad R_2 \quad \ldots \quad Q_{n-1} \quad R_{n-1} \quad Q_n \quad R_n$$

such that:

- *Each R_i is a response from an honest participant, and is of the form $\langle n_1, n_2, \ldots n_j, M_i \rangle$. Each $M_i \in \mathcal{A}$ is a message from the algebra and n_1, n_2,$\ldots n_j$ are the indices of previous elements of the sequence which were received or sent by that participant. (The first actions of a participant will contain no indicies.)*

- *Each $Q_i = \langle Q_{i,1}, Q_{i,2}, Q_{i,3}, \ldots Q_{i,j} \rangle$ is a sequence of adversary operations, where $Q_{i,k}$ has one of the following forms:*

  - $\langle$ *"new text"*, $M \rangle$ *where* $M \in \mathcal{T}$,
  - $\langle$ *"new nonce"*, $M \rangle$ *where* $M \in \mathcal{R}_{Adv}$,
  - $\langle$ *"new DH value"*, $M \rangle$ *where* $M \in \mathcal{D}_P$,
  - $\langle$ *"encryption key"*, $M \rangle$ *where* $M \in \mathcal{K}_{Pub}$,
  - $\langle$ *"decryption key"*, $M \rangle$ *where* $M \in \mathcal{K}_{Priv} \cap \mathcal{K}_{Adv}$,
  - $\langle$ *"signature key"*, $M \rangle$ *where* $M \in \mathcal{K}_{Sig}$,
  - $\langle$ *"verification key"*, $M \rangle$ *where* $M \in \mathcal{K}_{Ver} \cap \mathcal{K}_{Adv}$,
  - $\langle$ *"symmetric key"*, $M \rangle$ *where* $M \in \mathcal{K}_{Sym} \cap \mathcal{K}_{Adv}$,
  - $\langle$ *"concatenation"*, $n_1, n_2, M \rangle$ *where* $M = M_1\, M_2$ *and* M_1 *(resp.* M_2*) is the message of the sequence (or sub-sequence) element indexed by* n_1 *(resp.* n_2*),*
  - $\langle$ *"separation-left"*, $n_1, M \rangle$ *where* M' *is the message of the sequence or subsequence element indexed by* n_1 *and* $M' = M\, M''$ *for some* M'',
  - $\langle$ *"separation-right"*, $n_1, M \rangle$ *where* M' *is the message of the sequence or subsequence element indexed by* n_1 *and* $M' = M''\, M$ *for some* M'',
  - $\langle$ *"encryption"*, $n_1, n_2, M \rangle$ *where* $M = \{\!|M'|\!\}_K$ *and* M' *(resp* K*) is the message in the sequence or subsequence element indexed by* n_1 *(resp.* n_2*), and* $K \in \mathcal{K}_{Pub} \cup \mathcal{K}_{Sym}$,

- $\langle$ *"decryption"*, $n_1, n_2, M\rangle$ *where* $M' = \{\!|M|\!\}_K$ *for some* $K \in \mathcal{K}_{Pub} \cup \mathcal{K}_{Sym}$ *and* M' *(resp.* K^{-1}*) is the message in the sequence (or subsequence) element indexed by* n_1 *(resp* n_2*),*
- $\langle$ *"signing"*, $n_1, n_2, M\rangle$ *where* $M = [M']_K$ *for some* $K \in \mathcal{K}_{Ver}$ *and* M' *(resp.* K^{-1}*) is the message in the sequence (or subsequence) element indexed by* n_1 *(resp* n_2*),*
- $\langle$ *"message from sig"*, $n_1, M\rangle$ *where* $M' = [M]_K$ *for some* $K \in \mathcal{K}_{Ver}$ *and* M' *is the message in the sequence or subsequence element indexed by* n_1*,*
- $\langle$ *"hashing"*, $n_1, M\rangle$ *where* $M \in hash(M')$ *and* M' *is the message of the sequence (or sub-sequence) element indexed by* n_1*, or*
- $\langle$*"knownDH"*$, n_1, n_2, M\rangle$ *where* $M = DHd_1, d_2$*,* n_1 *references a node with message* d_1*, and* n_2 *references a node previous in the trace of the form* $\langle$ *"new DH value"*, $M\rangle$*,*
- $\langle$ *"function"*, $n_1, n_2, \ldots n_i, f, M\rangle$ *where*
  * M *is an element of* $\mathcal{D}$*,*
  * *each* M_i*, the message of node* n_i*, is an element of* $\mathcal{D}$*,*
  * f *is the description of a PPT-computable function, and*
  * *For all tape* $\leftarrow \{0,1\}^*$, $[\![F_f(M_1, M_2, \ldots M_i)]\!]^t_\eta = [\![M]\!]^t_\eta$.

In keeping with notation from other frameworks in formal cryptography [63, 62], we will denote by "node" the sequence elements of honest participants and the elements of adversary sub-sequences. We will call the last component of a node its "message."

It will soon be convenient to define the first point in a trace that uses elements from a set S:

Definition 38 *A set S* arises *at a node n of a trace iff n is the first node of the trace with a message in the set* $I = \{M' : \exists M \in S \text{ such that } M \sqsubseteq M'\}$.

That is, a set arises at the first node whose message has, as an ingredient, elements of the set. Such nodes are important because they indicate the genesis of certain elements. The node on which a random nonce arises, for example, indicates the point at which it was randomly selected and thus the entity that flipped the needed coins.

We will also require an analogous definition (originally from [63]) for the subterm relation:

Definition 39 *A set S* originates *at a node n of a trace iff n is the first node of the trace with a message in the set* $I = \{M' : \exists M \in S \text{ such that } M \prec M'\}$.

We also speak of a single message M arising or originating at a node n, by which we actually mean the set $\{M\}$.

hence, an origination point for a value v is the first place at which v could be learned or said by the adversary. The concept of origination points is usually used in the negative, meaning that we very often wish to prove that certain values do not originate at all. Note that the adversary is able to "say" (put in the trace) anything that it can deduce. Hence, one proves that a value v is secret by showing that no trace can contain that value. More specifically, if one can show that there is no point on which v originates, then there is no node which has that value as a subterm. (If there is at least one node with v as a subterm, then there must be a first instance, which would be an origination point.) Thus, to prove the non-origination of v is the Dolev-Yao way to prove secrecy of that value.

4.4 The Security Property $\mathcal{DH}$ and Conservative Protocols

In this section, we will define a condition on traces which represents the computational Diffie-Hellman assumption, and a condition on protocols which makes them efficient to execute.

The computational Diffie-Hellman assumption is relatively easy to embed in the expanded Dolev-Yao model. Informally, the computational Diffie-Hellman assumption states that if g^x and g^y are chosen by honest participants, then the adversary has only negligible probability of computing g^{xy}. In keeping with Section 4.3.1, we represent g^x by d_1 and g^y by d_2. We represent the fact that they are chosen by honest participants by having them arise on participant nodes. The message $DH d_1, d_2$ represents g^{xy}. We ignore negligible probabilities, and represent adversary computation via traces. Hence, the computational Diffie-Hellman assumption can be represented as:

Definition 40 (Security Property $\mathcal{DH}$) *Suppose that $\mathcal{T}$ is a valid, explicit Dolev-Yao trace over a protocol. The trace $\mathcal{T}$ satisfies condition $\mathcal{DH}$ if, for all d_a and $d_b \in \mathcal{D}$, whenever d_a and d_b arise on participant nodes in $\mathcal{T}$, then $DH(d_a, d_b)$ does not originate on an adversary node in $\mathcal{T}$.*

That is, if d_a and d_b arise on honest nodes, meaning that they are chosen by honest participants, then $DH(d_1, d_2)$ does not originate at all, meaning that in particular it is secret from the adversary.

This condition easily represents the inability of the adversary to solve the computational Diffie-Hellman problem on its own. Suppose a trace violates this condition but contains no non-trivial participant nodes. That is, assume that the trace contains only two participant nodes, $R_1 = \langle d_1 \rangle$ and $R_2 = \langle d_2 \rangle$, and that all other nodes in the trace are adversarial. Each adversary node represents one calculation, and each such calculation can be performed efficiently. The terms d_1, d_2, and $DH(d_1, d_2)$ represent g^x, g^y, and g^{xy} respectively, and so the trace represents an algorithm that takes in g^x and g^y from the participants and outputs g^{xy} at the node containing $DH(d_1, d_2)$. Note, also, all of this is independent of the exact cryptographic algorithms. Hence, if the Diffie-Hellman problem is hard, then there exists no trace that violates condition $\mathcal{DH}$—which does not also contain participant actions.

But what about traces that *do* involve participants? For these traces, condition $\mathcal{DH}$ may be too strong. There is no restriction on the actions of participants, after all, and so there is no prohibition against the protocol (for example):

On input d_1 and d_2, output $F_f(DH(d_1, d_2))$

where f is some easily invertible permutation on the underlying group. Although $DH(d_1, d_2)$ does not syntactically originate from this adversary, the protocol enables the adversary to compute it (by calculating f^{-1}. A trace involving this protocol could violate condition $\mathcal{DH}$, but the adversary does not actually violate the Diffie-Hellman assumption. There is no reason to assume that the Diffie-Hellman problem remains hard if the adversary has access to oracles (participants) that perform intractable calculations. Hence, there is no *a priori* reason that condition $\mathcal{DH}$ must hold for traces over arbitrary protocols.

Therefore, we restrict our attention to those protocols that do not provide assistance to the adversary. In general, this means that protocols must be simulatable. If the (computational) adversary can simulate the execution of the protocol, then honest participants give no assistance to the adversary. Any help they could give would be already available.

Rather than to consider all simulatable protocols, which may not be easy to characterize syntactically, we instead consider only a sub-class. There are many classes from which to choose; we choose ours based on such protocols as TLS and SSH. These protocols share a natural but important condition: participants do not "say" the key g^{xy}, but only hash it into key material.

Definition 41 (Silence) *We say that a protocol is* silent *with respect to Diffie-Hellman if no element of $\mathcal{D}_{DH}$ originates on a participant node.*

Here, we do mean "originate" and not "arise." The definition allows elements of $\mathcal{D}_{DH}$ to arise on participant nodes so long as they do not originate there. That is, a protocol is silent with respect to Diffie-Hellman if, whenever $DH(d_1, d_2)$ arises, it has been hashed into a symmetric key. This property is purely syntactic and easy to verify. It also allows us to prevent, in the following way, the honest participants from solving the computational Diffie-Hellman problem for the adversary.

In the next section, we are going to give a mapping from traces to algorithms. We will then show that a trace $\mathcal{T}$ which violates condition $\mathcal{DH}$ must map to an algorithm that solves the computational Diffie-Hellman problem. Interestingly enough, this result is independent of the exact computational algorithms used in the encoding operator $[\![\cdot]\!]^t_\eta$. Thus, $\mathcal{T}$ actually maps to an infinite number of algorithms—one for each choice of computational algorithms—and each one of them must solve the computational Diffie-Hellman problem.

These algorithms do not necessarily violate the computational Diffie-Hellman assumption, since the honest participants may not be efficient. Thus, even if they compute the Diffie-Hellman secret for the Dolev-Yao adversary, the running time of the joint system (participants and adversary) will still be larger than polynomial. If a protocol is silent, however, then we can prevent the honest participants from aiding the adversary in any way.

If a protocol is silent, then participants can only use $DH(d_1, d_2)$ as an ingredient of a symmetric key. This means that $DH(d_1, d_2)$ must be hashed before it is used, and this gives us a chance to prevent it from being "leaked" to the computational adversary. If the computational hash algorithm completely hides the pre-image, as the random oracle will, then the behavior of the honest participant will be completely independent of the value g^{xy}. By using the random oracle as the hash algorithm, we can map the trace $\mathcal{T}$ to an algorithm where the (bounded) adversary must still solve the computational Diffie-Hellman problem even though it receives no aid from the (unbounded) participants.

More formally:

Theorem 11 *Suppose that $\mathcal{T}$ is a valid, explicit Dolev-Yao trace (for $\{g_\eta, G_\eta\}_\eta$) over a silent protocol that violates condition $\mathcal{DH}$. Then the existence of random oracles implies the falsehood of the computational Diffie-Hellman assumption for $\{g_\eta, G_\eta\}_\eta$.*

We have used this theorem with great success to analyze real-world protocols. (See [32] for an analysis of TLS.) Here, however, we focus not on its usage but upon its computational justification, which we develop in the next two sections.

4.5 A Mapping from Traces to Algorithms

In this section, we provide two things:

1. A natural mapping from traces to computational algorithms that uses arbitrary computational sub-algorithms, and
2. A proof that for any such choice of algorithms, a trace which violates condition $\mathcal{DH}$ maps to an algorithm that solves the computational Diffie-Hellman problem.

The mapping will use the encoding operator $[\![\cdot]\!]_\eta^t$ in an essential way, so we must extend this operator to the remaining operations of the extended Dolev-Yao algebra:

Definition 42 (Extended encoding, II) *Let $\eta \in \mathcal{N}$ be the security parameter. Let $t \in \{0,1\}^\omega$ be a random tape, partitioned into a length-η segment for each element of $\mathcal{R}$, $\mathcal{K}$, and $\mathcal{D}$. Let $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be an asymmetric encryption scheme, $(\mathsf{G_{sym}}, \mathsf{E_{sym}}, \mathsf{D_{sym}})$ be a symmetric encryption scheme, $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$ be a digital signature scheme, and H be a hash scheme. Let $\{g_\eta, G_\eta\}_\eta$ be a family of cyclic groups. Then for any $M \in \mathcal{A}$, the* encoding *of M, written $[\![M]\!]_\eta^t$, is defined recursively:*

- *If $M \in \mathcal{D}$, M is of the form $DH(d_1, d_2)$, or M is of the form $F_f(d_1, d_2, \ldots d_n)$, then $[\![M]\!]_\eta^t$ is as in Definition 36.*
- *If $M \in \mathcal{R}$, $M \in \mathcal{K}_{Pub}$, $M \in \mathcal{K}_{Priv}$, $M \in \mathcal{M}$, M is of the form $M_1\,M_2$, or M is of the form $\{\![M']\!\}_K$ for some $K \in \mathcal{K}_{Pub}$, then $[\![M]\!]_\eta^t$ is as defined in Definition 5.*

- *If $M = hash\,(M')$, then $[\![M]\!]^t_\eta$ is the mapping from pairs of distributions to distributions given by $\left\langle \mathsf{G}_{\mathsf{sym}}\left(1^\eta, \mathsf{H}\left([\![M']\!]^t_\eta\right)\right), \text{“symkey”}\right\rangle$. Note that the hash output is used as randomness to generate a symmetric key.*

- *If (M, M^{-1}) is a signature/verification key pair, then $[\![M]\!]^t_\eta = \langle \mathsf{s}, \text{“sigkey”}\rangle$ and $[\![M^{-1}]\!]^t_\eta = \langle \mathsf{v}, \text{“verkey”}\rangle$ where (s, v) is the output of $\mathsf{G}_{sig}(1^\eta, \sigma_M)$.*

- *If $M = [M']_K$ is an encryption, then $[\![M]\!]^t_\eta$ is the mapping from pairs of distributions to distributions given by $\left\langle [\![M']\!]^t_\eta, \mathsf{S}_{sig}\left([\![M']\!]^t_\eta, [\![K]\!]^t_\eta\right), \text{“sig”}\right\rangle$*

- *If $M \in \mathcal{K}_{Sym}$ is a symmetric key, then $[\![M]\!]^t_\eta = \langle \mathsf{k}, \text{“symkey”}\rangle$ where k is the output of $\mathsf{G}_{\mathsf{sym}}(1^\eta, \sigma_M)$.*

- *If $M = \{\!|M'|\!\}_K$ is a symmetric encryption (meaning that $K \in \mathcal{K}_{Sym}$) then $[\![M]\!]^t_\eta$ is the mapping from pairs of distributions to distributions given by $\left\langle \mathsf{E}\left([\![M']\!]^t_\eta, [\![K]\!]^t_\eta\right), \text{“symenc”}\right\rangle$*

A word about how the mapping from traces to algorithms will proceed: The resulting algorithm will calculate (and store in a table) a value for each node and ingredient thereof in the trace. Very often, but not always, the value for a message M will be drawn from $[\![M]\!]^t_\eta$. The mapping itself assumes no properties about the cryptographic sub-algorithms, meaning we are free to choose these sub-algorithms arbitrarily.

Definition 43 *Let $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ be an asymmetric encryption scheme, $(\mathsf{G}_{\mathsf{sym}}, \mathsf{E}_{\mathsf{sym}}, \mathsf{D}_{\mathsf{sym}})$ be a symmetric encryption scheme, $(\mathsf{G}_{sig}, \mathsf{S}_{sig}, \mathsf{V}_{sig})$ be a digital signature scheme, and H be a hash scheme. Let $\{G_\eta\}_\eta$ be a family of cyclic groups. Let $\mathcal{T}$ be a trace over $\mathcal{A}$. Then $\mathtt{A}_{\mathcal{T}}(1^\eta)$ is the following algorithm:*

- *First, a table T is created to map elements of $\mathcal{A}$ to bit-strings.At the beginning of the execution, this table is empty.*

- *A tape t is randomly selected from $\{0,1\}^*$.*

- *Each node in the trace is then replaced with a bit-string value, starting with the first node and working forward. Let node n have message M. The exact manner in which a string is chosen for n depends on n's type:*

- *Suppose that n is of the form $\langle \text{“function”}, n_1, n_2, \ldots n_i, f, M\rangle$. Then:*
  - *f is a PPT-computable function, and*
  - *By inductive hypothesis, a value v_{n_j} has already been chosen for each node n_j.*

  The value v_n for n is chosen by running $\mathtt{M}_f(v_{n_1}, v_{n_2}, \ldots v_{n_i})$ and returning the output. Additionally, this is the value for $T(M)$ (i.e., stored in the table T for message M) if no value is already present there.

- *If n (containing message M) is any other kind of node, then there are two cases:*
  - *If $T(M)$ exists, then v_n is given the value of TM.*
  - *If $T(M)$ is not assigned (meaning that M is not in the table T) then a value $v_n \leftarrow [\![M]\!]_\eta^t$ is generated. Since the $[\![\cdot]\!]_\eta^t$ operator is defined recursively on the parse tree of M, the ingredients of M are drawn from or stored in the table T during this process as well.*

The above algorithm converts each node of the trace $\mathcal{T}$ into a bit-string. We now define what it means for it to have performed the conversion correctly:

Definition 44 *Let $\mathcal{T}$ be a trace and $\mathtt{A}_\mathcal{T}$ be the algorithm derived from $\mathcal{T}$ as per Definition 43. Then an execution of $\mathtt{A}_\mathcal{T}$ is "correct" if, given that t is the tape selected at the beginning of the execution and T is the table at the end, $T(M) \in [\![M]\!]_\eta^t$ for all M with entries in T.*

Theorem 12 *If $\mathcal{T}$ is a valid, explicit Dolev-Yao trace, then* $\Pr[\mathtt{A}_\mathcal{T}(\eta)$ *computes correctly* $]$ *is noticable.*

Proof. The only part of the algorithm what would store a value in $T(M)$ which is not in the support of $[\![M]\!]_\eta^t$ is that which processes "function" nodes. Let n be a node of the form

$$\langle \text{"function"}, n_1, n_2, \ldots n_i, f, M \rangle .$$

Then by induction, each n_j is a previous node with message M_j, and the probability

$$v_{n_j} \in \mathit{supp}\ [\![M_j]\!]_\eta^t \geq \frac{1}{q_j(\eta)}$$

for some polynomial q_j and sufficiently large η.

The value v_n is found by running $\mathtt{M}_f$ on v_1, v_2,$\ldots v_n$. By assumption

$$\Pr[\mathtt{M}_f(v_{n_1}, v_{n_2}, \ldots v_{n_i}) = f(v_{n_1}, v_{n_2}, \ldots v_{n_i})] \geq \frac{1}{q(\eta)}$$

for some polynomial q and all sufficiently large η. But notice that the distribution

$$f([\![M_1]\!]_\eta^t, [\![M_2]\!]_\eta^t, \ldots [\![M_i]\!]_\eta^t) = \left[\!\!\left[F_f([\![M_1]\!]_\eta^t, [\![M_2]\!]_\eta^t, \ldots [\![M_i]\!]_\eta^t) \right]\!\!\right]_\eta^t .$$

Thus

$$\begin{aligned}\Pr\left[f(v_{n_1}, v_{n_2}, \ldots v_{n_i}) \in \mathit{supp}\ \left[\!\!\left[F_f([\![M_1]\!]_\eta^t, [\![M_2]\!]_\eta^t, \ldots [\![M_i]\!]_\eta^t) \right]\!\!\right]_\eta^t \right] &\geq \prod_{j=1}^{i} \Pr\left[v_{n_j} \in \mathit{supp}\ [\![M_j]\!]_\eta^t \right] \\ &\geq \prod_{j=1}^{i} \frac{1}{q_j(\eta)}\end{aligned}$$

and

$$\Pr\left[a \leftarrow \mathtt{M}_f(v_{n_1}, v_{n_2} \ldots v_{n_i}) : a \in \mathit{supp}\ \left[\!\left[F_f(\llbracket M_1 \rrbracket^t_\eta, \llbracket M_2 \rrbracket^t_\eta, \ldots \llbracket M_i \rrbracket^t_\eta)\right]\!\right]^t_\eta\right] \geq \frac{1}{q(\eta)} \prod_{j=1}^{i} \frac{1}{q_j(\eta)}.$$

However, we know from the definition of valid traces that $\left[\!\left[F_f(\llbracket M_1 \rrbracket^t_\eta, \llbracket M_2 \rrbracket^t_\eta, \ldots \llbracket M_i \rrbracket^t_\eta)\right]\!\right]^t_\eta = \llbracket M \rrbracket^t_\eta$. Furthermore, the value $T(M)$ stored in the table is drawn from $\mathtt{M}_f(v_{n_1}, v_{n_2}, \ldots v_{n_i})$. Thus

$$\Pr\left[t \leftarrow \{0,1\}^\omega : T(M) \in \mathit{supp}\ \llbracket M \rrbracket^t_\eta\right] \geq \frac{1}{q(\eta)} \prod_{j=1}^{i} \frac{1}{q_j(\eta)}$$

for some all sufficiently large η.

Furthermore, a trace has only a constant number of "function" nodes, so the probability the algorithm calculates and stores the correct value for each remains noticeable over the entire execution of the algorithm. ■

Corollary 13 *Suppose $\mathtt{A}_\mathcal{T}$ correctly executes, and let T be the table at the end of the execution. For all d_1, $d_2 \in \mathcal{D}$, if $T(d_1) = g^x$ and $T(d_2) = g^y$, then $T(DH(d_1, d_2)) = g^{xy}$.*

Proof. If the algorithm correctly executes, then $T(M) \in \mathit{supp}\ \llbracket M \rrbracket^t_\eta$. Thus if $T(d_1)$ contains g^x, then $\llbracket d_1 \rrbracket^t_\eta = g^x$ with probability 1 (where t is the tape chosen by $\mathtt{A}_\mathcal{T}$. Similarly, $\llbracket (\rrbracket^t_\eta d_2) - g^y$ with probability 1. Hence, $\llbracket DH(d_1, d_2) \rrbracket^t_\eta = g^{xy}$ with probability 1, and $T(DH(d_1, d_2))$ will contain g^{xy}. ■

Hence, if a trace uses $DH(d_1, d_2)$ at any point, then the algorithm can be used to solve the Diffie-Hellman problem. Notice that the cryptographic algorithms are not used except to convert messages to bit-strings, so $\mathtt{A}_\mathcal{T}$ will work no matter how they are chosen. However, the algorithm $\mathtt{A}_\mathcal{T}$ may not be computable in PPT. As described, it requires that the computational Diffie-Hellman problem be solved for each term $DH(d_1, d_2)$ in the trace that doesn't arise on an f-node. In particular, if the trace contains an instance of $DH(d_1, d_2)$ that arises on a participant node, the resulting algorithm may be forced to solve the computational Diffie-Hellman problem. In the next section, we discuss a way around this difficulty.

4.6 Hashing and the Random Oracle

Assume that there exists a trace over a silent protocol which violates security property $\mathcal{DH}$. As shown in the previous section, this trace maps to an algorithm that solves the Diffie-Hellman problem no matter how the computational cryptographic algorithms are chosen for the encoding operation. However, the resulting algorithm may not be efficient. In particular, the algorithm assigns the value g^{xy} to the formal term $DH(d_1, d_2)$ (when it also assigns g^x to d_1 and g^y to d_2). This may require the algorithm to solve the computational Diffie-Hellman problem directly—an operation we are explicitly assuming to be inefficient.

However, all adversary nodes *are* efficiently computable; the algorithm would only need to solve Diffie-Hellman when calculating the values on participant nodes. Furthermore, we now assume that all of the participant nodes are silent, and hence Diffie-Hellman values are only used on participant nodes to make symmetric keys. Since we assume that making a key from a Diffie-Hellman value involves hashing it first, we can use this to avoid an infeasible computation.

The central idea is that, if the output of the hashing algorithm is completely independent of the input, then the honest participants can provide no aid to the adversary. More formally, the honest participants can be simulated by simply choosing random values for the hash algorithm to output—thus avoiding the need to calculate the pre-image, which might include the secret Diffie-Hellman value. For simplicity, we use the strongest possible hash algorithm: the random oracle. (We will later discuss the possibility of weaker constructions.)

Theorem 11 *Suppose that $\mathcal{T}$ is a valid, explicit Dolev-Yao trace (for $\{g_\eta, G_\eta\}_\eta$) over a silent protocol that violates condition $\mathcal{DH}$. Then the existence of random oracles implies the falsehood of the computational Diffie-Hellman assumption for $\{g_\eta, G_\eta\}_\eta$.*

Proof. By assumption, $\mathcal{T}$ is over a silent protocol and violates security property $\mathcal{DH}$. Then:

- d_1 and d_2 arise only on participant nodes in $\mathcal{T}$,
- $DH(d_1, d_2)$ does not originate on a participant node, but
- $DH(d_1, d_2)$ originates in $\mathcal{T}$.

Let n be the node on which $DH(d_1, d_2)$ originates. Note that n must be an adversary node. By inspecting the form of adversary nodes, we see that the message of n must be $DH(d_1, d_2)$ itself. (If the message contained $DH(d_1, d_2)$ as a subterm, then $DH(d_1, d_2)$ must be an subterm of a previous node and n would origination at n.)

Let $\mathcal{T}|_n$ be the set of all nodes in $\mathcal{T}$ which are before n. We construct an adversary $\mathtt{A}$ that breaks the computational Diffie-Hellman assumption in the following way: $\mathtt{A}(1^\eta, g, g^x, g^y)$ simulates $\mathtt{A}_{\mathcal{T}|_n}$, with the following important exceptions:

1. Instead of the table T being empty at initialization, it contains an entry mapping d_1 to g^x and an entry mapping d_2 to g^y.
2. When $\mathtt{A}_{\mathcal{T}|_n}$ calculates a value for $DH(d_a, d_b)$, it seems to need to solve the Diffie-Hellman problem in order to do so. However, we can avoid this calculation by considering the kinds of nodes which would cause $\mathtt{A}_{\mathcal{T}|_n}$ to calculate a value for $DH(d_a, d_b)$.

- It could be a f node, in which case $\mathtt{A}$ simulates $\mathtt{M}_f$ as $\mathtt{A}_{\mathcal{T}|_n}$ would.
- It could be an adversary node of the form $\langle$"$knownDH''$, $n_1, n_2, DH(d_1, d_2)\rangle$. In this case, however, $d_2 \in \mathcal{D}_P$, and so the exponent of $g^y = T(d_2)$ can be found on the segment of tape t assigned to d_2.
- If could be a participant node, in which case we know that $DH(d_a, d_b)$ is not a subterm of the node in question. (If it were, then it would have originated there.) Let the node in question contain message M. Since it is an ingredient but not a subterm of M, we can see that $DH(d_a, d_b) \sqsubseteq hash\,(N) \sqsubseteq M$ by examination of the term structure. Therefore, we only need to calculate a Diffie-Hellman value to hash it into a symmetric key.

  We employ a trick: instead of calculating the hash, we store a random value instead. That is, instead of calculating M normally, $\mathtt{A}$ chooses a random r of the proper length for $hash\,(N)$ and stores it in the table T. It does not calculate a value for N or any of its ingredients (including $DH(d_a, d_b)$) as part of this calculation.

When finished simulating $\mathtt{A}_{\mathcal{T}|_n}$, the adversary $\mathtt{A}$ selects the value g^z calculated for the node n and returns g^z as its output.

What is the likelihood that the new algorithm $\mathtt{A}$ will output the correct value?

Let us revisit the original algorithm $\mathtt{A}_{\mathcal{T}|_n}$. We know from Theorem 12 that for some polynomial q and all sufficiently large η:

$$\Pr\left[\mathtt{A}_{\mathcal{T}|_n}(1^\eta) \text{ computes properly}\right] \geq \frac{1}{q(\eta)}.$$

Note that we can modify $\mathtt{A}_{\mathcal{T}|_n}$ to take the values for d_1 and d_2 as inputs. In that case, running the new algorithm on random inputs is exactly the same as running it before the modifications:

$$\begin{array}{lll}\Pr[& x, y \leftarrow \{1, 2, \ldots, |G_\eta|\} : & \\ & \mathtt{A}_{\mathcal{T}|_n}(1^\eta, g^x, g^y) \text{ computes properly} &] \geq \frac{1}{q(\eta)}\end{array}$$

We can also modify $\mathtt{A}_{\mathcal{T}|_n}$ to output g^z, where $\langle g^z, \text{"DH value"}\rangle$ is the value computed for node n. Due to Corollary 13:

$$\begin{array}{lll}\Pr[& x, y \leftarrow \{1, 2, \ldots, |G_\eta|\}\,; & \\ & g^z \leftarrow \mathtt{A}_{\mathcal{T}|_n}(1^\eta, g, g^x, g^y) : & \\ & g^{xy} = g^z &] \geq \frac{1}{q(\eta)}\end{array}$$

That is, the original algorithm $\mathtt{A}_{\mathcal{T}|_n}$ can calculate the Diffie-Hellman value g^{xy} with some polynomial probability. But we are running $\mathtt{A}$, not $\mathtt{A}_{\mathcal{T}|_n}$. Will the new algorithm have the same advantage? $\mathtt{A}$ uses random values for hashing, while the original algorithm $\mathtt{A}_{\mathcal{T}|_n}$ calculates the values by application of the hash algorithm for $\mathtt{A}_{\mathcal{T}|_n}$. However, we can choose the hash algorithm arbitrarily, and so we can choose the random oracle. In this case, the hash values are random, and so this $\mathtt{A}$ has exactly the same probability of success as $\mathtt{A}_{\mathcal{T}|_n}$. ■

In other words, suppose that the random oracle exists. If there exists a valid, explicit Dolev-Yao trace over a silent protocol that violates the security property $\mathcal{DH}$ of Definition 40, the computational Diffie-Hellman assumption is false over the group family in question. Conversely, if the Diffie-Hellman problem is hard over the group and the random oracle is assumed, then there can be no silent and conservative trace which violates the security condition $\mathcal{DH}$.

Chapter 5

Related and Future Work

5.1 Computational Soundness for the Dolev-Yao Model

The first connection between the Dolev-Yao and the computational models was established in work by Abadi and Rogaway ([3, 4], and continued in [2] and [45, 46]) which served as a great source of inspiration for this work. In particular, these authors derived and implemented the symmetric-key version of Theorem 3. Our indistinguishability property is derived for the public-key encryption setting, and is both stronger than theirs in some places and weaker than theirs in others. More importantly, however, we go on to use our indistinguishability property to achieve non-malleability.

The relationship between indistinguishability and non-malleability depends on the setting (see [7] for an examination of this issue). In purely computational definitions of encryption security, for example, non-malleability always implies indistinguishability, but the converse is not always true. The results of this paper, interestingly, are exactly the opposite. We show that indistinguishability (Theorem 3) implies non-malleability (Definitions 10 and 12) in the Dolev-Yao setting. This serves as strong evidence that non-malleability implies indistinguishability, but the question is still open.

Another two related research efforts in this area are those of Backes, Pfitzmann and Waidner [6], and Micciancio and Warinschi [47]. In general, these investigations represent protocol executions in two different ways: a "real" setting and an "ideal" setting. In the "real" setting, the execution of a protocol is represented as the communication of Turing machines that use computational encryption to create bit-string messages. The two lines of research differ in their representation of the "ideal" setting. Backes et al. use a 'database' that stores all messages and tracks which ones are known by whom. This database allows the adversary to access only those messages it would be able to deduce in the Dolev-Yao paradigm. Micciancio and Warinschi, on the other hand, represent the ideal setting directly as symbolic execution in the Dolev-Yao model. The main results of both efforts state that any behavior that an honest participant can see in the "real" setting could also be seen in the "ideal" setting. Hence, a proof of security in the "ideal" setting will serve as a proof in the "real" setting (modulo negligible probabilities).

These works are extremely compelling. However, they focus attention onto the behavior of the adversary as a whole. That is, they regard the adversary's behavior as an unknowable mystery which cannot be broken into component parts. We, on the other hand, regard the behavior of the adversary as a series of message creations, and leverage a statement about a single creation into a statement about the adversary's behavior as a whole.

A less similar approach to the same problem is a recent effort to incorporate polynomial-time indistinguishability into process algebras [36, 37, 42, 48, 49]. Process algebras introduce grammars for processes that typically encompass a large number of higher-level programming constructs. They also introduce a number of algebraic rewrite and cancellation laws that allow one to prove two processes equivalent, that their observable behaviors are equivalent, or that the observable behavior of one process is a subset of the observable behavior of another. In this framework, one can prove a given process to be "safe" by showing that its observable behavior is the same as, or a subset of, the observable behavior of an idealized "specification" process.

This idea has recently been expanded to include new types of "equivalent" behavior. In particular, the definitions of both process and observable behavior have been expanded to include probabilistic behavior. This allows the definition of "observationally equivalent" to mean "indistinguishable to any polynomial-time environment or distinguisher." This approach does not provide the tools necessary to prove an original indistinguishability result, but it does allow one to prove that some given indistinguishability result follows from another one. Furthermore, this derivation uses the high-level rewrite and cancellation rules of the algebra rather than direct reductions.

5.2 Plaintext-Aware Encryption

Plaintext-awareness (the random-oracle form) was originally introduced as an intermediate definition in the chosen-ciphertext security proof of an asymmetric encryption scheme [10]. However, it was later shown to be strictly stronger than chosen-ciphertext security [7]. This same work also showed that plaintext-awareness implied chosen-ciphertext security. It further refined the original definition in much the same way that the ally oracle refines ours: by incorporating the adversary's access to externally-generated ciphertexts.

There have been previous efforts to remove the random-oracle assumption from the definition of plaintext-awareness [51]. However, these apply only to specific encryption schemes, and are not as strong as the original definition. We are, as far we know, the only work to propose such a radical redefinition of plaintext awareness.

5.3 Diffie-Hellman and the Dolev-Yao Model

We believe this to be the first effort to use the computational model to incorporate Diffie-Hellman into the formal model. Previous work on protocols that use Diffie-Hellman [56, 40] have assumed a strictly formal adversary. That is, they assume an adversary with an explicitly enumerated set of operations. Although the sets of operations chosen in these works include some appropriate to the Diffie-Hellman scheme, they do not consider the possibility of arbitrary efficient computations. Our work, on the other hand, assumes only what can be justified in terms of computational cryptography. Hence, proofs in our framework will be as strong as the Diffie-Hellman assumption. (Note, however, our work is not as widely applicable as that in [56]: we cannot yet consider common group-keying protocols, for example.)

However, our approach currently uses the random oracle. Of course, our original results on the standard Dolev-Yao model used the random oracle as well. We believe that the random oracle plays the same role in both instances: a technical simplification that bridges some purely computational difficulties. In the case of our original Dolev-Yao work, the random oracle (via the original definition of plaintext-awareness) allowed us to prove that computational soundness for Dolev-Yao non-malleability existed. This, in turn, allowed us to find a more sophisticated soundness that did not use the random oracle. Similarly, now that we have demonstrated one version of soundness for the Dolev-Yao Diffie-Hellman, we anticipate future work that remove the random oracle.

In particular, the random oracle was used here to make the hash of a secret independent of the secret itself. The fact that it is a *secret* being hashed indicates that the same result could be accomplished with a weaker construction. In particular, a pseudo-random function might work instead, although a chicken-and-egg problem could result if the seed for this function needs to be secret and independent. Another possibility is to extract the (limited) amount of computational entropy guaranteed by the computational Diffie-Hellman assumption. (See [13] for a discussion of entropy under this assumption. See also [23] for a study of a very similar problem.)

We trust that future work will examine this issue, and will be able to remove the random oracle from our results.

Chapter 6

Acknowledgements

I have no shortage of people to thank, beginning with my committee.

Nancy Lynch inspired me with her tireless curiosity. Many were the times that her perceptive questions revealed solutions to what had previously been difficulties.

Ron Rivest provided priceless guidance and advice throughout my tenure as his student. I am indebted to the wise counsel he dispensed most generously.

Silvio Micali, through his classes, taught me everything I know about cryptography. Furthermore, he taught me everything I presently know about what it means to do cryptographic research.

Both Silvio Micali and Moses Liskov allowed me the pleasure and honor authoring a paper with them. All the insight and originality to be found in Chapter 4 are theirs; all mistakes and oversights are my own.

Be Blackburn simultaneously served as an anchor to the "real" world of common sense while also providing unmeasurable amounts of assistance to those of us who had somehow lost outs. It is also likely that much of this document would not have been possible without her steady supply of cryptography fuel: chocolates.

My researches were supported by the Cambridge-MIT Institute and by NSF grant CCR-0326277.

It was my mentor at mentor at MITRE, Josh Guttman, who taught me how to do mathematical research, a lesson that has served me well in graduate school. It is the possibility of working with him again that makes a return to MITRE so appealing.

Lastly, none of this would have even been possible without the support and love of my wife, Amy Herzog, who is lying next to me as I write—and finish writing—these words.

Appendix A

Index of notation

A.1 The standard Dolev-Yao model

We use the following notation for sets and operations in the setting of formal cryptography:

$A \to B : M$	Entity A sends the message M, addressed to B. (No guarantee that it reaches B.)
M, N	Arbitrary elements of $\mathcal{A}$
$\{\|M\|\}_K$	Message M encrypted using key K.
K^{-1}	Function that maps an asymmetric key the other of the pair, and a symmetric key to itself
$M\,N$	The concatenation of messages M and N
$\mathcal{A}$	The formal algebra of terms
$\mathcal{T}$	The set of plaintexts
$\mathcal{M}$	The set of names
$\mathcal{R}$	The set of nonces
$\mathcal{K}$	The set of keys
$\mathcal{K}_{Pub}$	The set of public keys
$\mathcal{K}_{Priv}$	The set of private keys
$C[S]$	The closure of set S
$\mathcal{R}_{Adv}$	Nonces chosen by the adversary
$\mathcal{K}_{Adv}$	Private keys chosen adversary
$\mathcal{R} \setminus \mathcal{R}_{Adv}$	Nonces whose values are picked randomly by honest participants
$\mathcal{K}_{Subv}$	Keys of principals subverted by the adversary after their keys have been chosen
K_A	The public key of entity A
$\mathit{pattern}_{pk}(M, T)$	The public-key pattern of M Given the set of keys T.
$\mathcal{T}_M$	The type tree of M
$M \sqsubseteq N$	M is an ingredient of M
$M \prec N$	M is a subterm of M
$\mathcal{T}$	A Dolev-Yao trace
n	A node

A.2 Computational sets and operations

We use the following notation for sets and operations in the setting of computational cryptography:

η	Security parameter
$f \leq neg(\eta)$	The function f is negligible in the security parameter
$D_\eta \cong D'_\eta$	Distribution families D_η and D'_η are computationally indistinguishable
$D_\eta \cong_{\mathtt{O}} D'_\eta$	Distribution families D_η and D'_η are computationally indistinguishable with respect to oracle $\mathtt{O}$
$\mathcal{N}$	The natural numbers
$x \leftarrow D$	x is drawn from the distribution D. (If D is a set, x is drawn uniformly from D.)
$\Pr[a_1, a_2, \ldots : p]$	The probability that predicate p is true after experiments a_1, $a_2 \ldots$
G	The key generation algorithm for public-key encryption
E	The encryption algorithm for public-key encryption
D	The decryption algorithm for public-key encryption
$\mathsf{G}_{\mathsf{sym}}$, $\mathsf{E}_{\mathsf{sym}}$, $\mathsf{D}_{\mathsf{sym}}$	The key generation, encryption and decryption algorithms (respectively) for symmetric encryption
G_{sig}, S_{sig}, V_{sig}	Signature key generation, signature and verification algorithms (respectively)
H	The hash evaluation algorithm
e	A computational encryption key
d	A computational decryption key
s	A computational signature key
v	A computational verification key
k	A computational symmetric key
c	A computational ciphertext
sig	A signature
$\mathsf{Parameter}$	The set of security parameters (typically $\mathcal{N}$)
Coins	The set of random tapes (coin flips)
$\mathsf{PublicKey}$	The set of possible public keys
$\mathsf{PrivateKey}$	The set of possible private keys
$\mathsf{Ciphertext}$	The set of possible encryptions
$\mathsf{Plaintext}$	The set of possible plaintexts
String	$\{0,1\}^*$, the set of finite bit strings
$\mathtt{A}$	The adversary
$D_1 \cong D_2$	Distributions D_1 and D_2 are computationally indistinguishable
$\mathtt{P}$	The prover of an interactive proof system
$\mathtt{V}$	The verifier of an interactive proof system
$\mathtt{S}$	The simulator
σ	The reference string for a NIZK
$\mathtt{E}$	The extractor
$\mathtt{O}$	An oracle
$\mathcal{NP}$	The complexity class
PPT	The complexity class
$supp\, D$	The support of probability distribution D
$\mathsf{TRANS}^{A,B}(x, r_A \vert y, r_B)$	Transcript of protocol between A and B
$\mathsf{VIEW}_A^{A,B}(x, r_A \vert y, r_B)$	View of A during protocol with B
$\mathsf{OUT}_A^{A,B}(x, r_A \vert y, r_B)$	Output of A after protocol with B
G, G_η	Group over which Diffie-Hellman exchange is performed
g	Generator of Diffie-Hellman group

A.3 Expansions to the Dolev-Yao model

$\mathcal{A}_\eta$	The finite version of the $\mathcal{A}$
$\mathcal{R}_\eta$	The finite version of $\mathcal{R}$
$\mathcal{K}_{Pub\eta}$	The finite version of $\mathcal{K}_{Pub}$
$\mathcal{K}_{Priv\eta}$	The finite version of $\mathcal{K}_{Priv}$
$[M]_K$	Message M with a signature verifiable with key K.
$\mathcal{D}$	Set of formal Diffie-Hellman values
DH	Formal Diffie-Hellman operation
$\mathcal{D}_{DH}$	Set of Diffie-Hellman values which result from the Diffie-Hellman operation (range of DH)
$\mathcal{D}_P$	Set of formal Diffie-Hellman values generated by the adversary (subset of $\mathcal{D}$)
$hash$	Hash operation
F_f	Dolev-Yao operator to represent function f
$\mathcal{K}_{Sig}$	The set of signing keys
$\mathcal{K}_{Ver}$	The set of verification keys
$\mathcal{K}_{Sym}$	The set of symmetric keys

A.4 Sets and operations used to connect formal and computational settings

We use the following notation for sets and operations used to bridge and connect the formal and computational settings:

$[\![M]\!]_\eta^t$	The distribution of M, induced by running the `Convert` algorithm on M
μ	Function from formal name to bit-strings
t	The tape used to generate atomic terms
$t\,()$	Function from expressions to tags
$vis_\tau\,(\sigma)$	String σ is visible in string τ
v_n	Value given to node in algorithm $\mathtt{A}_\mathcal{T}$
T	Table of algorithm $\mathtt{A}_\mathcal{T}$
$\mathtt{M}_f$	Machine to evaluate function f

A.5 Mathematics

$x \leftarrow D$	x is drawn from distribution D
$x \leftarrow S$	x is drawn uniformly from distribution D
$\forall$ s.l. η	For all sufficiently large η ($\exists \eta_0$ s. t. $\forall \eta \geq \eta_0$)
$\lvert x \rvert$	Bit-length of x

A.6 Plaintext-Awareness

$h \xleftarrow{H} \mathtt{A}$	Extract current history from $\mathtt{A}$
RU	Registration user
RA	Registration authority
$\mathtt{L}$	Ally oracle
s	Private input for ally

Bibliography

[1] *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)*. IEEE Computer Society, June 1999.

[2] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In Naoki Kobayashi and Benjamin C. Pierce, editors, *Proceedings, 4th International Symposium on Theoretical Aspects of Computer Software TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 82–94. Springer, 2001.

[3] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, August 2000.

[4] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[5] Manindra Agarwal, Nitin Saxena, and Neeraj Kayal. PRIMES is in P. Available at `http://www.cse.iitk.ac.in/news/primality.html`, 2003.

[6] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proceedings, 10th ACM conference on computer and communications security (CCS)*, October 2003. Full version available at `http://eprint.iacr.org/2003/015/`.

[7] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Krawczyk [34], pages 26–45. Full version found at `http://www.cs.ucsd.edu/users/mihir/papers/relations.html`.

[8] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Brickell [15], pages 390–420.

[9] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology - CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, August 1993. Full version of paper available at `http://www-cse.ucsd.edu/users/mihir/`.

[10] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption– how to encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt 94 Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.

[11] M. Blum, P. Feldman, and S. Micali. Non-interactive zero knowledge proof systems and applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 103–112, 1988.

[12] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.

[13] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Advances (CRYPTO 96)*, volume 1109 of *Lecture Notes in Computer Science*, pages 129–142. Springer-Verlag, August 1996.

[14] Joan Boyar, Ivan Damgård, and René Peralta. Short non-interactive cryptographic proofs. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(4):449–472, 2000.

[15] Ernest F. Brickell, editor. *Advances in Cryptology (CRYPTO 1992)*, volume 740 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[16] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th ACM Symp. on Theory of Computing (STOC)*, pages 209–218, May 1998.

[17] R Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Krawczyk [34], pages 13–25.

[18] T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999.

[19] T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999.

[20] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[21] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of Advances of Cryptology (CRYPTO 84)*, number 196 in Lecture Noted in Computer Science, 1984.

[22] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *Advances in Cryptology– EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer-Verlag, May 1999.

[23] Rosario Gennaro, Hugo Krawczyk, and Tal Rabin. Secure hashed Diffie-Hellman over non-DDH groups. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 361–381. Springer, May 2004.

[24] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[25] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[26] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowedge complexity of interactive proof systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 291–304, 1985. Superseded by journal version.

[27] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital-signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[28] Shafi Goldwasser and Rafail Ostrovsky. Invariant signatures and non-interactive zero-knowledge proof are equivelent. In Brickell [15], pages 228–244.

[29] Shafi Goldwasser and Yael Taumann. On the (in)security of the Fiat-Shamir paradigm. In *Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, October 2003.

[30] Jonathan Herzog. Computational soundness for formal adversaries. Master's thesis, Massachusetts Institute of Technology, October 2002.

[31] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer-Verlag, August 2003.

[32] Jonathan C. Herzog. The Diffie-Hellman key-agreement scheme in the Strand-Space model. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 234–247. IEEE Computer Society, June 2003.

[33] J. Kohl and C. Neuman. The Kerberos network authentication service (v5). RFC 1510, September 1993.

[34] H. Krawczyk, editor. *Advances in Cryptology - CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, August 1998.

[35] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. Request For Comments (RFC) 2104, February 1997. Available at `http://www.cs.ucsd.edu/users/mihir/papers/rfc2104.txt`.

[36] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communication Security (CCS '98)*, pages 112–121, November 1998.

[37] P. D. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. Probabilistic polynomial-time equivalence and security protocols. In Jeannette M. Wing, Jim Woodcock, and Jim Davies, editors, *World Congress on Formal Methods*, volume 1708 of *Lecture Notes in Computer Science*, pages 776–793. Springer, September 1999.

[38] Gavin Lowe. An attack on the Needham–Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[39] Gavin Lowe. Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer–Verlag, 1996.

[40] Nancy Lynch. I/O automaton models and proofs for shared-key communication systems. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)* [1], pages 14–29.

[41] Anna Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Massachusetts Institute of Technology, September 2002.

[42] P. Mateus, J.C. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In Roberto M. Amadio and Denis Lugiez, editors, *Proceedings, 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 323–345. Springer, 2003.

[43] Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.

[44] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130, 1999.

[45] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. Workshop on Issues in the Theory of Security (WITS '02), January 2002.

[46] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004.

[47] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings, Theory of Cryptography Conference*, number 2951 in Lecture Notes in Computer Science, pages 133–151. Springer, February 2004.

[48] J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). In *Proc. 17th Annual Conference on the Mathematical Foundations of Programming Semantics (MFPS 2001)*, volume 45 of *Electronic Notes in Theoretical Computer Science*, May 2001.

[49] J. C. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *39th Annual Syposium on Foundations of Computer Science (FOCS 1998)*, pages 725–733. IEEE Computer Society, November 1998.

[50] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murφ. In *Proceedings, 1997 IEEE Symposium on Security and Privacy*, pages 141–153. IEEE, Computer Society Press of the IEEE, 1997.

[51] Siguna Müller. On the security of a Williams based public key encryption scheme. In Kwangjo Kim, editor, *4th International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001)*, volume 1992 of *Lecture Notes in Computer Science*. Springer, February 2001.

[52] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 427–437, May 1990.

[53] Roger Needham and Michael Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[54] National Institute of Standards and Technology (NIST). Secure hash standard. Federal Information Processing Standards (FIPS) Publication 180-1, April 1995. Available at `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[55] L C Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[56] Olivier Pereira and Jean-Jacques Quisquater. A security analysis of the cliques protocols suites. In *14th IEEE Computer Security Foundations Workshop — CSFW'01*, pages 73–81, Cape Breton, Canada, 11–13 June 2001. IEEE Computer Society Press.

[57] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and the chosen-ciphertext attack. In *Advances in Cryptology– CRYPTO 91*, Lecture Notes in Computer Science, pages 433–444.

[58] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[59] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th Annual Syposium on Foundations of Computer Science (FOCS 1999)*, pages 543–553. IEEE Computer Society, October 1999.

[60] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO 1988*, number 293 in Lecture Notes in Computer Science, pages 52–72. Springer-Verlag, 1988.

[61] D. Song. Athena, an automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW 12)* [1], pages 192–202.

[62] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[63] F. Javier THAYER Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[64] T. Ylonen, T. Kivinen, and M. Saarinen. SSH protocol architecture. Internet draft, November 1997. Also named draft-ietf-secsh-architecture-01.txt.