On Symbolic Analysis of Cryptographic Protocols

by

Akshay Patil

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

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2005

Author . . . .
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by . . . .
Ronald L. Rivest
Viterbi Professor of Computer Science
Thesis Supervisor

Certified by . . . .
Ran Canetti
Visiting Scientist
Thesis Supervisor

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

On Symbolic Analysis of Cryptographic Protocols

by

Akshay Patil

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

The universally composable symbolic analysis (UCSA) framework layers Dolev-Yao style symbolic analysis on top of the universally composable (UC) secure framework to construct computationally sound proofs of cryptographic protocol security. The original proposal of the UCSA framework by Canetti and Herzog (2004) focused on protocols that only use public key encryption to achieve 2-party mutual authentication or key exchange. This thesis expands the framework to include protocols that use digital signatures as well.

In the process of expanding the framework, we identify a flaw in the framework's use of UC ideal functionality F_{PKE}. We also identify issues that arise when combining F_{PKE} with the current formulation of ideal signature functionality F_{SIG}. Motivated by these discoveries, we redefine the F_{PKE} and F_{SIG} functionalities appropriately.

Thesis Supervisor: Ronald L. Rivest
Title: Viterbi Professor of Computer Science

Thesis Supervisor: Ran Canetti
Title: Visiting Scientist

Acknowledgments

My heartfelt thanks to my advisor Ron Rivest for his guidance and confidence in me. Working with him has been a constant inspiration to achieve my very best.

To Ran Canetti for always answering my questions, even though he knows I'm lying when I say they'll only take a moment. His attention to detail and willingness to listen have been life-saving parachutes on the many occasions I've blindly charged forward in my research.

To Jonathan Herzog for first introducing me to the UCSA framework. I would still be utterly mystified by formal analysis if not for his ability to communicate clearly and concisely.

To Be Blackburn, Maria Sensale, and Paula Mickevich for mothering me while I've been 3000 miles away from home.

To all the friends I've made during my years at MIT; their companionship has made my time here truly enjoyable.

To the NSF and MIT for making this research possible – I was supported by NSF grant CCR-032677 and the views expressed in this thesis are mine, not those of the NSF or MIT.

And to my family, whose love and support make everything possible.

Contents

List of Figures

Chapter 1

Introduction

A *cryptographic protocol* is a set sequence of messages exchanged between two or more parties who are trying to accomplish something. This "something" is what we call the protocol's *goal* – perhaps it's the ability to establish a shared secret or maybe it's just the reassurance that a certain someone is still alive. Whatever the goal, the ability to accomplish it, despite the presence of some adversary bent on being evil, is what we call *protocol security.*

Proving protocol security is difficult. There are a number of things that can go wrong on a number of different levels. In general, we have models and tools that help us analyze different parts of the protocol at different level, but no one way of combining these analysis results into single, convincing proof of protocol security

In [16], Ran Canetti and Jonathan Herzog proposed combining two seemingly complimentary bodies of work in order to create a unified framework for proving the security of protocols. They layered the *Dolev-Yao model* [22] on top of the *universally composable (UC) secure framework* [13] in order to create a new framework they named *universally composable symbolic analysis (UCSA).* This framework takes a protocol and its goal, then justifiably abstracts the protocol to a form where automated tools can help generate a convincing proof that the protocol is secure, i.e. achieves its goal even in the presence of an adversary.

Canetti and Herzog constructed universally composable symbolic analysis for protocols that only use public key encryption. In this thesis we expand the framework to also include digital signatures. In doing so, we identify and repair a flaw in their construction as well as redefine the ideal public key encryption and digital signature functionalities to include some new and needed properties of interest.

The parts of this thesis Chapter 2 provides a brief explanations of the models and security definitions that are used in the universally composable symbolic analysis framework. Chapter 3 then walks through Canetti and Herzog's construction of the framework. Chapter 4 identifies a flaw in part of their construction and motivates the need for new definitions of UC functionalities F_{PKE} and F_{SIG}. Chapter 5 presents and explains the new definitions, followed by Chapter 6 which analyzes the computational security needed to realize them. Finally, we once more present the universally composable symbolic analysis framework in Chapter 7, complete with our fixes and the addition of digital signatures. Chapter 8 concludes with suggestions for related future research.

Chapter 2

Preliminaries

This chapter explains the models used to construct the UCSA framework as well as some security definitions that are important for understanding the results of the framework

2.1 Computational Cryptography

Computational cryptography treats cryptography as a branch of complexity theory, modeling adversaries as probabilistic polynomial-time (PPT) algorithms and defining security in terms of an adversary's probability in accomplishing certain tasks. Probabilistic polynomial time means that the adversary can make random choices, but must finish running in an amount of time that is polynomial with respect to the size of the input it is running on. Both the time-efficiency and the success probability of the adversary are generally tied to a number known as the *security parameter*, k, which is set by the environment.

Computational proofs are grounded in a class of computational complexity problems widely believed to be "hard," i.e. not solvable by a probabilistic algorithm in time polynomial to k. Examples of hard problems are the discrete log problem and factoring the product of two large primes. A cryptographic scheme's security is proved by assuming the scheme isn't secure, then constructing an efficient reduction to a hard problem, thus reaching a believed contradiction.

Because of their mathematical foundation, computational proofs are desired when proving cryptographic security. These proofs, however, are hard to construct when trying to analyze protocols. In addition to potentially unforeseen interactions between the different primitives used, there are often security concerns when multiple instances of a protocol are being run simultaneously. While it is sometimes possible to provide a computational proof of protocol security, doing so is difficult and requires significant effort on the part of the prover.

2.2 Public Key Encryption and Digital Signature Security Definitions

In this section we define and explain the intuition behind computational notions of security that will be useful later.

2.2.1 Public Key Encryption

An encryption scheme is a system allowing a party to transmit a message to another party without an adversary learning the contents (or any function) of the message. With public key encryption, this is achieved through the use of public and private keys. Before the transmission of a message, a party B publishes a public key K_B to all other parties. If party A wishes to securely send a message m to B, A transmits the ciphertext of m encrypted with K_B, $\{\!|m|\!\}_{K_B}$. Upon receiving $\{\!|m|\!\}_{K_B}$, B uses a private key K_B^{-1} that is known only to him, to decrypt the ciphertext and retrieve m. An adversary who knows K_B but not K_B^{-1} should not be able to extract any meaningful information about m when given $\{\!|m|\!\}_{K_B}$.

More formally, we define a public key encryption scheme as follows:

Definition 1 (Public Key Encryption) *A public key encryption scheme consists of three algorithms* (gen, enc, dec)*:*

- $gen : 1^k \rightarrow \mathcal{K} \times \mathcal{K}'$, *is a key generation algorithm.* $\mathcal{K}$ *and* $\mathcal{K}'$ *are the sets of possible public and private keys, respectively.*
- $enc : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{C}$, *is an encryption algorithm.* $\mathcal{M}$ *and* $\mathcal{C}$ *are the sets of possible plaintext messages and ciphertexts, respectively.*
- $dec : \mathcal{K}' \times \mathcal{C} \rightarrow \mathcal{M}$, *is a decryption algorithm.*

One of the strongest and common definitions of public key encryption is indistinguishability against adaptive chosen ciphertext attack (IND-CCA2). With encryption, an adversary should be totally clueless about the contents of a ciphertext. For IND-CCA2, we try to capture this intuition by saying that even if an adversary picks two messages and is given the ciphertext for one of them, it still can't guess which of the two messages is the plaintext.

As with many security definitions in computational cryptography, this type of security is described as a game played between the adversary and a challenger (the challenger is how we model what the adversary sees in the environment). As mentioned before, the adversary is a probabilistic algorithm that runs in time polynomial to the security parameter k.

Indistinguishable under chosen ciphertext attack, 2-phase (IND-CCA2)

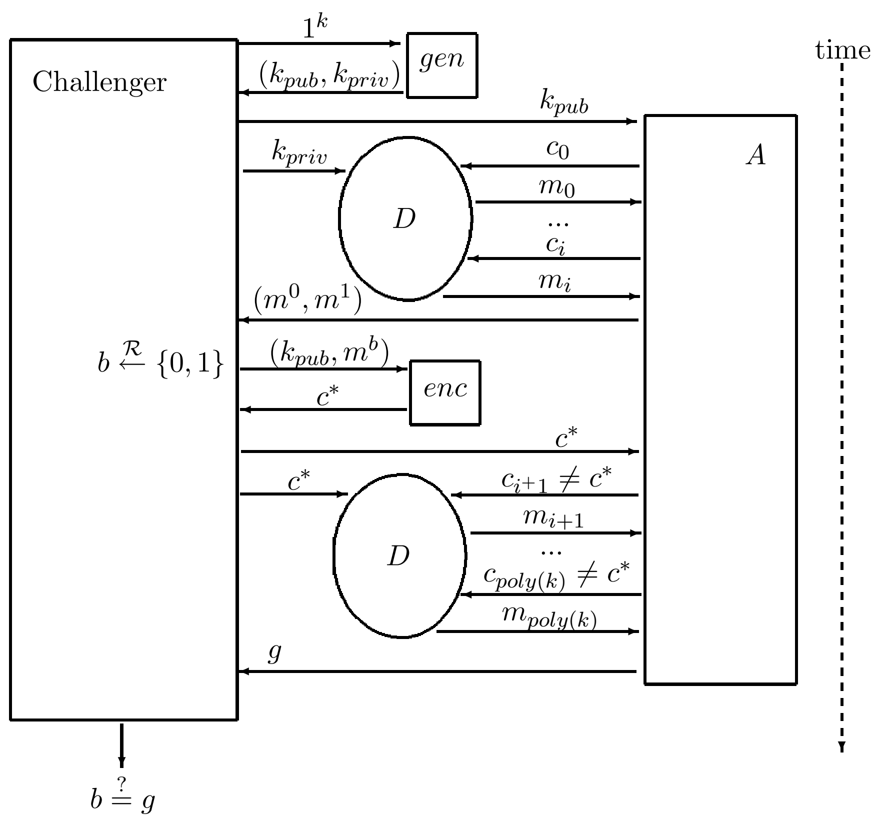

Figure 2-1: The IND-CCA2 game

For IND-CCA2, the game progresses as follows (Figure 2-1). The challenger runs $(k_{pub}, k_{priv}) \leftarrow gen(1^k)$ and hands k_{pub} to A, the adversary. The challenger then gives k_{priv} to a decryption oracle D. A can (adaptively) query D with ciphertexts c_i and D will send back the corresponding plaintexts $m_i \leftarrow dec(k_{priv}, c_i)$. After some number of queries, polynomial with respect to k, A hands the challenger two messages m_0 and m_1. The challenger picks b randomly from $\{0, 1\}$ and sets $c^* \leftarrow enc(k_{pub}, m_b)$; it then hands c to both D and A. A may now resume querying D, but is not allowed to ask for the decryption of c^*.

After a polynomial number of queries, the A outputs a guess g. If $g = b$, the adversary wins. A scheme is considered IND-CCA2 secure if no adversary wins with probability significantly better than it could achieve by just guessing a random g to begin with. More formally:

Definition 2 (IND-CCA2) *A public key encryption scheme consisting of three algorithms, $S = (gen, enc, dec)$, is called indistinguishable against adaptive chosen ciphertext attack if the following properties hold for any negligible function $neg()$ and all large enough values of the security parameter k with corresponding message space M_k:*

Completeness: *For all $m \in M_k$,*

$$\begin{aligned}\Pr[\quad & (k_{pub}, k_{priv}) \leftarrow gen(1^k); \\ & c \leftarrow enc(k_{pub}, m); \\ & m = dec(k_{priv}, c) \qquad\qquad] > 1 - neg(k)\end{aligned}$$

Ciphertext indistinguishability: *For any PPT adversary A with access to decryption oracle D,*

$$\begin{aligned}\Pr[\quad & (k_{pub}, k_{priv}) \leftarrow gen(1^k); \\ & (m_0, m_1) \leftarrow A^{D(k_{priv}, \cdot)}(k_{pub}, 1^k); \\ & b \stackrel{\mathcal{R}}{\leftarrow} \{0,1\}\,; \\ & c^* = enc(k_{pub}, m_b) \\ & g \leftarrow A^{D(k_{priv}, c^*, \cdot)}(k_{pub}, c^*, 1^k) : \\ & b = g \qquad\qquad\qquad\qquad] < \tfrac{1}{2} + neg(k)\end{aligned}$$

The extra completeness property ensures that a message remains the same after undergoing encryption and decryption. We define negligible functions as follows:

Definition 3 (Negligible Functions) *A function $f : \mathbb{N} \rightarrow \mathbb{R}$ is* negligible *in k if, for any polynomial q, $f(k) \leq \frac{1}{q(k)}$ for all sufficiently large k. If f is negligible in k, we write $f \leq neg(k)$.*

2.2.2 Digital Signatures

Digital signatures are used when a party wants to indicate they have originated a message m. A digital signature σ is dependent on some secret known only to the signer and on the message being signed. The recipient of (m, σ) can then run a public verification algorithm that will confirm or reject the message's signature without requiring knowledge of the signer's secret. In this thesis, we will restrict our attention to schemes that accomplish this using signing and verification keys.

We also restrict our attention to party behavior or signature schemes that somehow bind signatures to their messages – this may be through the use of message revealing signatures (where one can derive m from σ) or by treating a non-message revealing signature as invalid unless explicitly attached to a particular message. For clarity of expression, we adopt the latter practice and will always pair signatures with their messages in our examples.

More formally, we define a digital signature scheme as follows:

Definition 4 (Digital Signatures) *A digital signature scheme consists of three algorithms* (gen, sig, ver)*:*

- $gen : 1^k \to \mathcal{K} \times \mathcal{K}'$*, is a key generation algorithm.* $\mathcal{K}$ *and* $\mathcal{K}'$ *are the sets of possible signing and verification keys, respectively.*

- $sig : \mathcal{K} \times \mathcal{M} \to \mathcal{S}$*, is a signing algorithm.* $\mathcal{M}$ *and* $\mathcal{C}$ *are the sets of possible messages and signatures, respectively.*

- $ver : \mathcal{K}' \times \mathcal{M} \times \mathcal{S} \to \mathcal{B}$*, is a verification algorithm. It outputs a boolean bit indicating whether it accepts or rejects the message signature pair.*

A common definition for signature security is existential unforgeability under adaptive chosen message attack (EUF-ACMA). As with IND-CCA2, we can describe the security condition as a game between the adversary and the environment/challenger (Figure 2-2).

The challenger runs $(k_s, k_v) \leftarrow gen(1^k)$ and hands verification key k_v to A, the adversary. The challenger then gives k_s to a signing oracle S. A can (adaptively) query S with messages m_i and receive back the corresponding signatures $\sigma_i \leftarrow sig(k_s, m_i)$. After some polynomial number of queries, the adversary outputs a new message m that was never submitted as a query to S, and a signature forgery σ. If $ver(k_v, m, \sigma) = 1$, then the adversary wins. A scheme is considered EUF-ACMA if no adversary is able to win with non-negligible probability. We call the interaction between the adversary and the signing oracle an adaptive chosen message attack (ACMA)

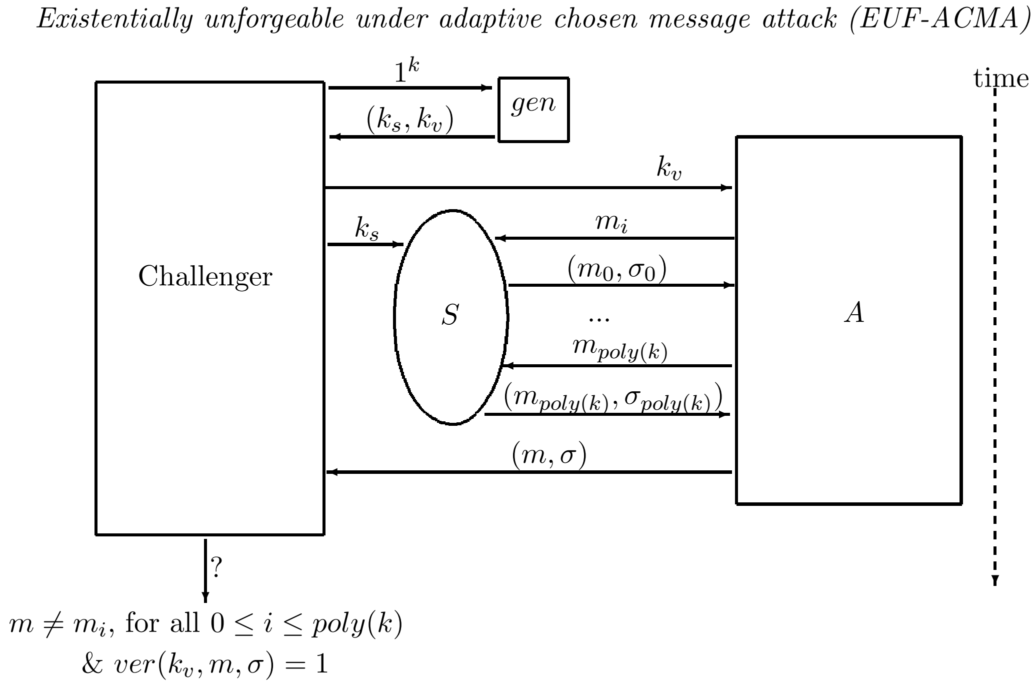

Figure 2-2: The EUF-ACMA game

Definition 5 (EUF-ACMA) *A signature scheme consisting of three algorithms, $\Sigma =$ (gen,sig,ver), is called existentially unforgeable under adaptive chosen message attack (EUF-ACMA) if the following properties hold for any negligible function neg() and all large enough values of the security parameter k with corresponding message space M_k:*

Completeness: *For all $m \in M_k$,*

$$\Pr[\quad (k_s, k_v) \leftarrow gen(1^k); \\ \sigma \leftarrow sig(s, m); \\ 0 \leftarrow ver(m, \sigma, v) \quad] < neg(k)$$

.

Unforgeability: *For any PPT forger A with access to signing oracle S,*

$$\Pr[\quad (k_s, k_v) \leftarrow gen(1^k); \\ (m, \sigma) \leftarrow A^{S(k_s, \cdot)} : \\ 1 \leftarrow ver(m, \sigma, k_v) \text{ and } A \text{ never asked } S \text{ to sign } m \quad] < neg(k)$$

EUF-ACMA security allows for multiple valid signatures on the same message – there are some situations that require a stronger definition of security. For example, there are non-malleable unique signature schemes, which are schemes resilient to an ACMA with only one valid signature for a given message and verification key. Even stricter are one-time signature schemes, schemes that only sign one message once for a given verification key.

There is, however, a security definition that is stronger than EUF-ACMA but weaker than unique signatures. Schemes that meet this alternative security definition are called "strong" [31, 5] [1] or "super-secure" [23]. Usage of strong signatures are not as common as EUF-ACMA or one-time signatures, but there are situations where one-time signatures are used when only strong signatures are necessary [21].

Strong security is very similar to EUF-ACMA – in fact, their definitions differ only slightly. In the definition of EUF-ACMA, the adversary is required to output a valid message/signature pair (m, σ) where m was never a query to the signing oracle. For strong security, we make the adversary's game easier: in order to win, it must output a valid pair (m, σ), where (m, σ) itself was never a response from the oracle. Referring back to the EUF-ACMA game in Figure 2-2, the strong signature game is the same as replacing the second-to-last line with "$(m, \sigma) \neq (m_i, \sigma_i)$ for all $0 \leq i \leq poly(k)$." We consider this to be an easier game since any adversary that wins the EUF-ACMA game wins the strong signature game as well. By making the game easier, we make our security definition stronger. A strong signature scheme not only guarantees that past signatures don't help for forging signatures on new messages, but that they also don't help for forging new signatures on old messages.

Definition 6 (Strong Signature Schemes) *A signature scheme consisting of three algorithms, Σ = (gen,sig,ver), is called* strong *if the following properties hold for any negligible function $neg()$ and all large enough values of the security parameter k with corresponding message space M_k:*

Completeness: *For any $m \in M_k$,*

$$\Pr[\quad (s, v) \leftarrow gen(1^k); \quad \sigma \leftarrow sig(s, m); \quad 0 \leftarrow ver(m, \sigma, v) \quad] < neg(k)$$

.

[1] This is a different definition of "secure signatures" than the one presented by Goldwasser, Micali, and Yao in 1983 [24] which was eventually renamed EUF-ACMA security

Strict Unforgeability: *For any PPT forger F with access to signing oracle S,*

$$\begin{aligned}\Pr[\quad & (s,v) \leftarrow gen(1^k);\\ & (m,\sigma) \leftarrow F^{S(s,\cdot)}:\\ & 1 \leftarrow ver(m,\sigma,v) \textit{ and } F \textit{ never received } (m,\sigma) \textit{ from } S \quad] < neg(k)\end{aligned}$$

2.3 Formal Cryptography

Formal cryptography uses mathematical or logical systems to analyze protocols. The results of such analysis, however, do not directly apply to real protocols because of the high level of abstraction used. For the purpose of this thesis, we will focus on symbolic analysis and the Dolev-Yao model. We first describe symbolic analysis and its importance in general terms, before describing the specifics of the Dolev-Yao model.

2.3.1 Symbols and Rules

In symbolic analysis, we think of messages as symbols and cryptographic operations as symbolic operations on these messages. For example, if we had a message that we represented with the symbol m, then the ciphertext of m encrypted under a public key K would be the symbol $\{\!|m|\!\}_K$, where we let $\{\!|\cdot|\!\}$ represent encryption. If we have an entity, we can represent the entity's initial knowledge as a set S of symbols. We can then write rules governing the way new symbols are formed and handled based on what the entity knows already. Given this initial set S, we can talk about the *closure* of S (denoted as $C[S]$) which is the result of applying these rules. Continuing with public key encryption, we might write the following rules:

1. $S \subseteq C[S]$
2. If $m \in C[S]$ and $K \in C[S]$, then $\{\!|m|\!\}_K \in C[S]$
3. If $\{\!|m|\!\}_K \in C[S]$ and $K^{-1} \in C[S]$, then $m \in C[S]$ (where K^{-1} is K's corresponding secret key)

Rule 1 says that anything the entity knows, it can derive. Rule 2 says that if the entity can derive a message and a public key, then the entity can derive the ciphertext of the message under the key. Lastly, Rule 3 says that if the entity can derive a ciphertext under a particular public key and can also derive the corresponding private key, then the entity can derive the ciphertext's plaintext.

We generally think of the formal model adversary as being in complete control of the network it is a part of. The ability to represent the knowledge of an adversary during a protocol run is important for this allows an analyzer to determine what messages an adversary is capable of creating when trying to attack the protocol. The closure operation does this, allowing a protocol checker to iterate all plausible attacks.

2.3.2 The Dolev-Yao Model

The Dolev-Yao model is a popular model for symbolic analysis which serves as the foundation for a number of formal methods. There are many variants on the Dolev-Yao model and we define one that serves our purpose of analyzing public key encryption. We also include the notion of a local output as proposed in [16].

The first thing to define is the set of atomic symbols from which the Dolev-Yao algebra is constructed.

Definition 7 (The Dolev-Yao Message Algebra) *Messages in the Dolev-Yao algebra $\mathcal{A}$ are composed of atomic elements of the following types:*

- *Party identifiers ($\mathcal{M}$) – These are denoted by symbols $P_1, P_2, ..$ for a finite number of names in the algebra. These are public and are associated with a* role *of either Initiator or Responder.*
- *Nonces ($\mathcal{R}$) – These can be thought of as a finite number of private, unpredictable random-strings. These symbols are denoted by $R_1, R_2, ...$ and so on.*
- *Public keys ($\mathcal{K}_{Pub}$) – These are denoted by symbols $K^e_{P_1}, K^e_{P_2}, ...$ which are public and each associated with a particular party identifier.*
- *A garbage term, written $\mathcal{G}$, to represent ill-formed messages,*
- *$\perp$, to represent an error or failure,*
- *Starting, to indicate that a protocol execution has begun, and*
- *Finished, to indicate that a protocol execution has ended.*

Messages in the algebra can be compounded by the following symbolic operations:

- *pair:* $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$. *When messages* m *and* m' *are paired, we write* $m|m'$.
- *encrypt* : $\mathcal{K}_{Pub} \times \mathcal{A} \rightarrow \mathcal{A}$. *When message* m *is encrypted with public key* K_P^e, *we write* $\{\!|m|\!\}_{K_P^e}$

.

The Dolev-Yao algebra is *free*, meaning that no two distinct symbols represent the same message. An important consequence of this is that we can define a *parse tree* for each message which describes the unique symbolic structure of the message (see Figure 2-3).

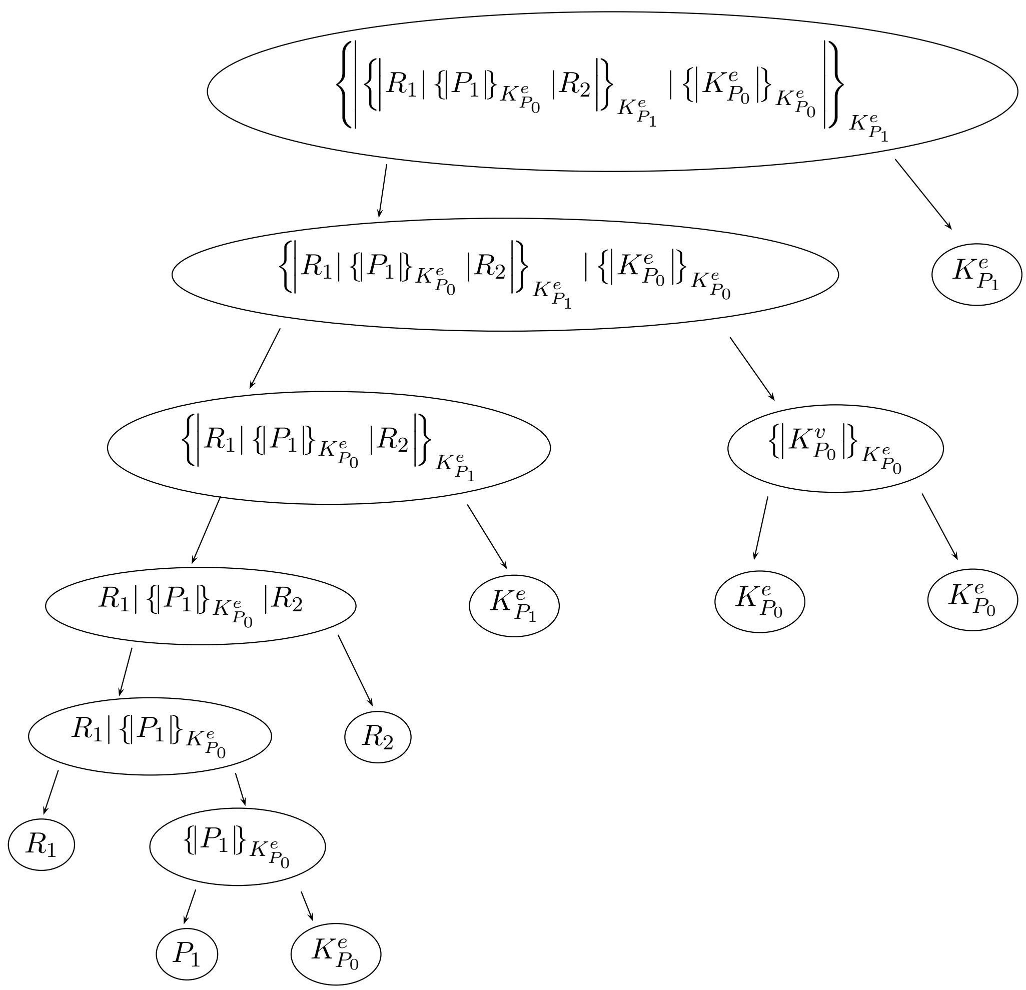

Figure 2-3: Example Dolev-Yao parse tree

As mentioned in Section 2.3.1, one of the major benefits of describing the world using symbols is that we can enumerate the messages it is possible for an entity to make, based on the symbols it already knows. In particular, we assume that the adversary completely controls the network, but that any message delivered by the adversary must be derivable from the adversary's initial knowledge and the messages communicated by honest parties.

The adversary's initial knowledge consists of all public keys ($\mathcal{K}_{Pub}$), all identifiers ($\mathcal{M}$), and those nonces that the adversary itself generates ($\mathcal{R}_{Adv}$). To derive new messages, the adversary has only a few symbolic operations available to it: pairing two known elements, separating a pair, encrypting with public keys, and decrypting messages whose keys belong to corrupted parties ($\mathcal{M}_{Adv} \subset \mathcal{M}$).

This restriction on the adversary's ability to derive messages is captured by the Dolev-Yao closure operation:

Definition 8 (Closure) *Let*

- $\mathcal{R}_{Adv} \subset \mathcal{R}$ *be the set of nonces associated with the adversary,*
- $\mathcal{K}^e_{Adv} = \{K^e_P : P \in \mathcal{M}_{Adv}\}$ *be the set of encryption keys belonging to corrupted parties* ($\mathcal{K}^e_{Adv} \subset \mathcal{K}_{Pub}$)*, and*

Then the closure of a set $S \in \mathcal{A}$*, written* $C[S]$*, is the smallest subset of* $\mathcal{A}$ *such that:*

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

Symbolic Protocols

We now define what it means for a party to engage in a symbolic protocol. A party running a protocol in the Dolev-Yao model consists of the following components:

- An identity (represented by a symbol from $\mathcal{M}$).
- A role in the protocol (either *Initiator* or *Responder*)
- An internal state – we represent the party's internal state as all messages received in execution so far.
- An input port from which the party receives its initial input and state
- A communications port that the party uses to send and receive messages in $\mathcal{A}$.
- A local output port where the party can output elements in $\mathcal{A}$.

A protocol in the Dolev-Yao model is then defined as a set of actions performed by parties, based on their identity, role, internal state, and an incoming message.

Definition 9 (Symbolic Protocol) *A symbolic protocol $\mathcal{P}$ is a mapping from states ($\mathcal{S} = \mathcal{A}^*$), identities ($\mathcal{M}$), roles ($\mathcal{O}$), and messages ($\mathcal{A}$, the incoming message) to a new state ($\mathcal{S}$, the old state plus the new message) plus an algebra value sent as a message ($\mathcal{A} \times message$) and/or a value given as a local output ($\mathcal{A} \times output$). More formally, $\mathcal{P}$ is:*

$$\mathcal{P} : \mathcal{S} \times \mathcal{M} \times \mathcal{O} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathcal{A} \times message \times \mathcal{A} \times output$$

When a party terminates, it is set to a special state that only transitions to itself and only outputs $\perp$.

Technically there are no constraints on the messages that parties are asked to output – this means we can define a symbolic protocol where a party is asked to output a message that is not derivable from its internal state. These protocols are not particularly useful, so we will only consider protocols that are derived from efficiently implementable protocols (see Section 3.2). Intuitively, if we think of a closure operation $C'[S]$ which is defined in terms of an honest party instead of the adversary, we can think of these good protocols as ones where a party's outputs during protocol execution must be within the closure of the party's internal state.

Dolev-Yao protocol trace

When a symbolic protocol is executed, the participants start with an internal state, then respond to messages delivered by the adversary who can produce any message in the closure of its knowledge (which includes all messages communicated by participants). When describing a protocol execution, we want to capture this initial state and the messages exchanged, as well as the internal operations performed by the adversary to produce messages not explicitly communicated by honest participants. We call this description the *Dolev-Yao trace* of the protocol execution.

Definition 10 (Dolev-Yao Trace) *We inductively define a Dolev-Yao trace t for protocol $\mathcal{P}$ as a description of events that occur during the execution of $\mathcal{P}$.*

$$t = H_0 \ H_1 \ H_2 \ ... \ H_{n-2} \ H_{n_1} \ H_n$$

where event H_i is either

- *of the form $[$"input"$, P, o_i, P', S]$, which indicates the initial input of participant P to take the role o_i and interact with participant P', assuming initial internal state S.*
- *an adversary event (where $j, k < i$) of the form*
  - *$[$"enc"$, j, k, m_i]$, in which case $m_k \in \mathcal{K}_{Pub}$ and $m_i = \{\!|m_j|\!\}_{m_k}$,*
  - *$[$"dec"$, j, k, m_i]$, in which case $m_k \in \mathcal{K}^e_{Adv}$, and $m_j = \{\!|m_i|\!\}_{m_k}$,*
  - *$[$"pair"$, j, k, m_i]$, in which case $m_i = m_j|m_k$*
  - *$[$"extract-l"$, j, m_i]$, in which case $m_j = m_i|m_k$ for some $m_k \in \mathcal{A}$,*
  - *$[$"extract-r"$, j, m_i]$, in which case $m_j = m_k|m_i$ for some $m_k \in \mathcal{A}$,*
  - *$[$"random"$, m_i]$, in which case $m_i = R$ for some $R \in \mathcal{R}_{Adv}$,*
  - *$[$"name"$, m_i]$, in which case $m_i = A$ for some $A \in \mathcal{M}$,*
  - *$[$"pubkey"$, m_i]$, in which case $m_i = K$ for some $K \in \mathcal{K}_{Pub}$,*
  - *$[$"deliver"$, j, P_i]$, in which case the message m_j is delivered to party P_i.*
- *or a participant event of the form $[$"output"$, P_i, m_i]$ or $[$"message"$, P_i, m_i]$, in which case $[$"deliver"$, k, p_i]$ is the most recent adversary event in the trace (for some k) and*

the protocol action for P_i in its current role and internal state, upon receiving m_k, is to output/send message m_i. ($\mathcal{P}(S_j, o_i, m_k, P_i) \rightarrow (S_i, m_i, \{output, message\})$).

2.3.3 Example: Needham-Schroeder Protocol

The classic example for illustrating the need and power of formal models is the Needham-Schroeder protocol (Figure 2-4). Formally, we can define this protocol as the symbolic

$$\begin{array}{rc} A \rightarrow B: & \{\!|A, R_A|\!\}_{K_B^e} \\ A \leftarrow B: & \{\!|R_A, R_B|\!\}_{K_A^e} \\ A \rightarrow B: & \{\!|R_B|\!\}_{K_B^e} \end{array}$$

Figure 2-4: The Needham-Schroeder protocol (informally)

protocol $\mathcal{P}_{NS}$ seen in Figure 2-5.

The purpose of this protocol is two-fold. Firstly, it functions as a way to exchange secret numbers – the two parties engaged in the protocol should be the only ones to learn the values R_A and R_B. Secondly, the protocol attempts to provide authentication: at the end of a successful protocol run, both parties should know each other's identity, that they are engaged in this protocol with each other, and what the values of R_A and R_B are.

This protocol was proposed in 1978 [30]. Seventeen years later, Lowe discovered and repaired a flaw in the protocol's satisfaction of the authentication conditions [26]. In Lowe's attack (Figure 2-6), A begins a protocol run with a malicious entity M. M then, posing as A, begins a protocol run with B and convinces B that it is interacting with A.

At the end of this attack, B believes it is interacting with A but A believes it is interacting with M. Lowe's fix was to include the responder's name in the second message, making it

$$A \leftarrow B : \{\!|B, R_A, R_B|\!\}_{K_A}$$

Lowe then used a model checker to prove that no other attacks of a similar nature were possible [27]. His attack illustrates the types of vulnerabilities that might be present in a protocol but may be difficult to discover unless the underlying implementations of cryptographic primitives are abstracted out.

Let $\mathcal{D}_P$ represent the initial input/state of party P, let $*$ denote a wildcard which can be used to match anything, and let P be a place holder for the party identity each party thinks it is engaged with. We define $\mathcal{P}_{NS}$ to be the mappings:

- $\{\mathcal{D}_A\} \times A \times \mathit{Initiator} \times \{\} \rightarrow$

$$\{\mathcal{D}_A \cup \{R_A\}\} \times \left\langle \{\!|A|R_A|\!\}_{K_P^e} \right\rangle \times \mathit{message} \times \langle \mathit{Starting}|A|P|K_P^e \rangle \times \mathit{output}$$

- $\{\mathcal{D}_A \cup \{R_A\}\} \times A \times \mathit{Initiator} \times \left\langle \{\!|R_A|R_P|\!\}_{K_A^e} \right\rangle \rightarrow$

$$S^{\perp} \times \left\langle \{\!|R_P|\!\}_{K_P^e} \right\rangle \times \mathit{message} \times \langle \mathit{Finished}|A|P|K_P^e \rangle \times \mathit{output}$$

- $\{\mathcal{D}_B\} \times B \times \mathit{Responder} \times \left\langle \{\!|P|R_P|\!\}_{K_B^e} \right\rangle \rightarrow$

$$\{\mathcal{D}_B \cup \{R_P, R_B\}\} \times \left\langle \{\!|R_P|R_B|\!\}_{K_P^e} \right\rangle \times \mathit{message} \times \langle \mathit{Starting}|B|P|K_P^e \rangle \times \mathit{output}$$

- $\{\mathcal{D}_B \cup \{R_P, R_B\}\} \times B \times \mathit{Responder} \times \left\langle \{\!|R_B|\!\}_{K_P^e} \right\rangle \rightarrow$

$$S^{\perp} \times \langle \mathit{Finished}|B|P|K_B^e \rangle \times \mathit{output}$$

- $S^{\perp} \times * \times * \times \langle * \rangle \rightarrow$

$$S^{*} \times \perp \times \mathit{output}$$

Figure 2-5: The Needham-Schroeder symbolic protocol, $\mathcal{P}_{NS}$

2.3.4 Strengths and Weaknesses of Symbolic Analysis

As the Section 2.3.3 example illustrates, abstracting cryptographic primitives to intuitive symbolic operators allows high level analysis which can identify protocol vulnerabilities that might be missed otherwise. An important byproduct of these formal analysis methods is the ability to create protocol checkers which can automate the task of searching for vulnerabilities (e.g. [9, 32, 29]).

While useful, formal method proofs are not rigorous. Symbolic analysis ignores the details of how cryptographic primitives are realized or information the adversary may learn outside of applying the closure operations. For example, most symbolic analysis tools do not include exponentiation as a closure operation. Thus, if a protocol involves the transmitting of values g and a, then uses the value g^a, the model would not capture the fact that the

$$
\begin{array}{ll}
A \to M : \{\!|A, R_A|\!\}_{K_M^e} & \\
 & M(A) \to B : \{\!|A, R_A|\!\}_{K_B^e} \\
 & M(A) \leftarrow B : \{\!|R_A, R_B|\!\}_{K_A^e} \\
A \leftarrow M : \{\!|R_A, R_B|\!\}_{K_A^e} & \\
A \to M : \{\!|R_B|\!\}_{K_M^e} & \\
 & M(A) \to B : \{\!|R_B|\!\}_{K_B^e}
\end{array}
$$

Figure 2-6: Attacking the Needham-Schroeder protocol

adversary also knows g^a. It is difficult to introduce exponentiation to these models since many protocols rely on exponentiation properties that violate the freeness of the model's algebra (e.g. commutativity, $(g^a)^b = (g^b)^a$).

In addition, the way primitives are used makes implicit assumptions about the security of the concrete schemes implementing these cryptographic primitives. Looking back on our symbolic rules for public key encryption, it is not hard to see that a scheme that realizes encryption as the model uses it must be quite strong, *at least* IND-CCA2 secure (indistinguishable against chosen ciphertext attack, 2 phases. See Section 2.2.1). Unless a scheme with proper security characteristics is used when implementing these protocols, proofs given in a formal model are incomplete at best.

By removing the details of cryptographic primitives, formal cryptography gives protocol designers insight into protocol vulnerabilities that would be difficult to see otherwise. This abstraction, however, is made with few or no restrictions or justifications, weakening any security guarantees one might wish to assert about the protocol.

2.4 The Universally Composable Security Framework

The universally composable (UC) secure framework [13] creates a model for analyzing protocol security by replacing cryptographic primitives and protocol goals with ideal functionalities. Here we present an informal overview of the UC framework, focusing on concepts that will be relevant for later sections of this thesis; we first describe how parties and protocols are modeled in the framework, then how these ideal functionalities work. Finally we explain what it means for a protocol to securely realize an ideal functionality.

2.4.1 Parties and Protocols

Parties in the UC framework are modeled by sets of interactive Turing machines (ITMs). All ITMs are required to run in PPT (as defined in Section 2.1) with respect to security parameter k. Each ITM represents a program running within a party, the programs communicate with other programs within the party using local input and output tapes. Each ITM has a session-identifier (SID) identifying the session or protocol instance it is participating in. Each ITM also has a party identifier (PID) identifying which party the ITM is a part of. Each ITM's identifier pair (SID,PID) is unique to the ITM. In addition, each party has a role identifier (RID) which identifies the party's role in a protocol as either initiator or responder.

ITMs have incoming and outgoing communication tapes which model the messages sent in and out of the network. The adversary itself is also an ITM with control over message delivery between parties, subject to the synchrony guarantee. Within the set of parties there are two types: corrupted parties and uncorrupted (or honest) parties. In the general UC framework, the adversary is able to adaptively corrupt honest parties – in this work, however, we will limit ourselves to non-adaptive adversaries which are not allowed to corrupt new parties during protocol execution.

A real world protocol is modeled as parties running the protocol in the presence of an adversary and environment ITM Z, with input z. The parties, environment, and the adversary are the protocol participants, all with the same security parameter k. The modeled protocol execution progresses as a sequence of activations of individual participants. Different participants must abide by different rules when activated, but while activated, a participant may read the appropriate tapes and write on the tape of at most one other participant. Once an activation is complete, the participant whose tape was modified is activated next – if no communication tapes were modified, then the environment Z is activated next. The following is a list of rules regulating each participant's behavior:

1. The environment is the first participant to be activated. The environment may read the local output tapes of all participants. It may then activate another party to run the protocol or write on the local input tape of a party or the adversary.

2. The adversary may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message (from some other party) on the incoming

communication tape of a party or report information to Z by writing on its own output tape.

3. A party reads its input (either from the environment or the adversary) and writes either an output on its local output tape or an outgoing message on its outgoing communication tape. If the party is honest, it follows its code; if the party is corrupted, the adversary is allowed to control the internal actions of the party.

Protocol execution ends with the halting of the environment Z, which outputs a single bit. We designate this output as $\text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ when Z is interacting with parties running protocol p in the presence of adversary A on security parameter k, input z, and participant randomness $\vec{r} = r_{\mathsf{Z}}, r_{\mathsf{A}}, r_1, r_2, \ldots$. We let $\text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)$ be a random variable representing $\text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ where $\vec{r}$ is chosen at random. Let $\text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}$ be the probability ensemble $\{\text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$.

An important point to observe is that all participants are only able to read their own local input and incoming communication tapes. This means communications from the environment or adversary to participants are not visible to other participants.

2.4.2 Ideal Functionalities, Ideal Protocols, and Hybrid Protocols

Ideal functionalities are descriptions of how various functionalities or tasks should behave. They are meant to capture our intuitive sense of what it means to do something like public key encryption or mutual authentication. An ideal functionality F is modeled by an ITM which interacts with the protocol parties and adversary, but not the environment. For convenience, we will refer to the ITM running F as F. We represent the interaction between an ideal functionality F and participants with a special type of protocol, called an *ideal protocol*, denoted I_{F}. In the ideal world, the ideal protocol for functionality F is to just have participants give their inputs to F and accept its outputs. This exchange between a party P and F is not visible to other participants,[2] but when P receives a value from F, it immediately copies the value to its local output tape.

As will become clearer later, an adversary's interaction with the ideal functionality differs from that of the protocol parties. The adversary that interacts with the ideal protocols is

[2] This is achieved by parties writing directly onto F's input tape and F responding onto their input tape

called the *simulator* (S). We also use $\text{IDEAL}_{\mathsf{F},\mathsf{S},\mathsf{Z}}$ to represent the probability ensemble $\{\text{EXEC}_{I_{\mathsf{F}},\mathsf{S},\mathsf{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

A *hybrid protocol* is a protocol where parties execute as described before, but have access to one or more copies of each ideal functionality. Instead of all parties and all protocol instances interacting with the same ideal functionality, the parties may interact with any number of ideal functionalities, distinguished by their unique SIDs. In the hybrid model, parties no longer automatically copy the results of their ideal functionality to their output tape.

2.4.3 Realizing Ideal Functionalities

We now have a protocol model where primitives have been replaced with ideal functionalities which behave precisely how we expect them to – the question is then whether it's possible for a concrete cryptographic schemes to realize these ideal functionalities. Moreover, what does it mean for a concrete scheme to realize an ideal functionality to begin with?

Intuitively, we want to say a scheme in the real world realizes an ideal functionality if running a protocol using the concrete scheme does not result in anything that couldn't have happened in the ideal world. To formalize this notion, we use the environment to assert that the results of the two worlds look the same.

In order to formalize, we need to define our notion of "looks the same." As you recall, we defined the environment Z to output a single bit at the end of a protocol execution – if the two worlds are *indistinguishable*, then all environments Z should be unable to act differently (output different bits) when interacting with the two worlds.

Definition 11 (Binary indistinguishability) *Two* binary *distribution ensembles* $\{X(k,a)\}_{k\in\mathbb{N},a\in\{0,1\}^*}$ *and* $\{Y(k,a)\}_{k\in\mathbb{N},a\ in\{0,1\}^*}$ *are called* indistinguishable *(written $X \approx Y$) if for any $c \in \mathbb{N}$ there exists $k_0 \in \mathbb{N}$ such that for all $k > k_0$ and for all a we have*

$$|\Pr(X(k,a)=1) - \Pr(Y(k,a)=1)| < neg(k)\,.$$

Definition 12 (Secure Realization of Ideal Functionality) *Let F be an ideal functionality and let p be a protocol. We say p* securely realizes *F if there exists a S such that for any environment Z we have*

$$\text{IDEAL}_{\mathsf{F},\mathsf{S},\mathsf{Z}} \approx \text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}$$

This definition says that a protocol p realizes F if for every adversary A wrecking havoc in the real world with p there is some simulator S that could have done the same thing in the ideal world with F (except in negligibly small instances of adversarial randomness).

The universal composition theorem

Say r is a protocol that securely realizes an ideal functionality F and p is some F-hybrid protocol. We can construct the composed protocol p^{r} where parties running p replace each copy of F with a new instance of r with fresh randomness. If r is a G-hybrid protocol (i.e. protocol r uses ideal functionality G), then p^{r} is a G-hybrid protocol as well. The universal composition theorem says that it doesn't matter if the protocol realizing one ideal functionality is itself a hybrid protocol – the environment Z is still unable to distinguish between protocol executions in the ideal world with these ideal functionalities and the real world using concrete schemes. [13].

Theorem 1 (Universal Composition) *Let F, G be ideal functionalities. Let p be a F-hybrid protocol and let r be a protocol that securely realizes F. Then, for any adversary A, there exists a simulator S such that for any environment Z,*

$$\text{EXEC}^{\mathsf{F}}_{\mathsf{p},\mathsf{S},\mathsf{Z}} \approx \text{EXEC}_{\mathsf{p}^{\mathsf{r}},\mathsf{A},\mathsf{Z}}$$

In particular, if p securely realizes functionality G, then so does p^{r}.

This theorem has three important implications:

1. It implies that it is sufficient to analyze the security of a single instance of a protocol. If a single instance of the protocol securely realizes an ideal functionality, then it will do so even when combined with other instances of itself or other protocols

2. The realization of ideal functionality G may be a hybrid protocol using other functionalities F. In the context of this thesis, this will be important for showing that a protocol achieve its goal:

Take a protocol goal and describe it as an ideal functionality G. If we want to know whether or not a concrete protocol p realizes G, then we can analyze if the F-hybrid version of p, with idealized cryptographic primitives, realizes G. If it does, then that implies that the concrete p realizes its goals when instantiated with cryptographic schemes that realize the ideal primitive functionalities.

3. In order for a concrete schemes to securely realize ideal functionality, each scheme must be reinitialized with new randomness for every new instance of the protocol. This is a rather onerous requirement since it means all parties must create and distribute new keys at the beginning of each protocol instance, dramatically increasing protocol overhead.

   We can, however, remove this requirement by (effectively) including the protocol instance SID in each ciphertext (for schemes realizing F_{PKE}) scheme or signature (for F_{SIG}, described later) as shown in previous work of universal composition with joint state [19].

Chapter 3

Universally Composable Symbolic Analysis for Public Key Encryption

In 1981, Dolev and Yao proposed their symbolic model for analyzing protocols [22]. In it, the analyzer thinks of cryptographic primitives as symbolic operations where everything behaves in an intuitive manner. For example, symbols are unambiguous – a nonce looks very different from an encryption key which looks different from an entity's name. If a message is encrypted, an entity without the proper key is unable to learn anything about the message's ciphertext: message length, (potentially) which key encrypted it, whether the plaintext is the same as another ciphertext's, etc. These sorts of assumptions are very strong and are not trivially satisfiable by most cryptographic schemes. Their model became the foundation for a number of model variants and computational tools, but it wasn't until recently that serious attention was given to reconciling the assumptions of the Dolev-Yao model with computational cryptography.

In 2000, Abadi and Rogaway first offered a way to meaningfully describe the computational security definitions needed to satisfy the behavioral assumptions of cryptographic primitives in the Dolev-Yao model [3, 4]. Subsequent works [2, 28, 25] have refined the technique of Abadi and Rogaway. The idea is to convert the symbols of the Dolev Yao model into bit strings using a concrete scheme for the cryptographic primitive in question. The goal of such a proof is to show that a real world adversary is unable to produce anything the symbolic adversary is unable to produce. This is done by considering an adversary that, interacting with the translated bit-string version of a Dolev-Yao exchange, outputs a string corresponding to the translation of a symbol not in the closure of symbolic adversary. Using the stated security of the scheme used in translation, it is shown that the probability of this happening, taken over the randomness of the translation and the adversary, is negligible for all adversaries. This implies real world adversaries interacting with strings using this scheme are no more powerful than their symbolic counterparts and we say that the scheme realizes the Dolev-Yao security assumptions for that primitive.

Developing the techniques for translating Dolev-Yao assumptions to computational security definitions was an important first step for creating computationally sound proofs of protocol security, but it still left some doubt. A concrete scheme may realize the Dolev-Yao notion of a primitive, but that still didn't prove that a protocol deemed "secure" in the Dolev-Yao model necessarily achieved a desired computational goal. In addition, the symbolic protocol checkers needed to check the security of protocols in the context of multiple protocol instances with potential composing issues – perhaps giving further constraints not originally considered when formulating the realization proof.

With these issues in mind, Ran Canetti and Jonathan Herzog proposed a new framework for leveraging symbolic analysis of protocols to construct computational proofs of security [16]. Symbolic analysis is effective at automating proof security but suffers from composing issues. The universally composable framework has simplified composing issues but still involves relatively detailed cryptography when attempting to analyze protocols. Canetti and Herzog layer Dolev-Yao analysis on top of the UC framework in order to capture the complementary benefits of both. This chapter describes their construction which is defined over mutual authentication protocols using public key encryption. In Chapter 7, we will reconstruct the framework, but for protocols that use both public key encryption and digital signatures.

The definitions presented in this chapter are taken directly from Canetti and Herzog's paper ([16]) with minor modifications for clarity.

3.1 Overview

The general idea of the universally composable symbolic analysis framework is to translate a real world protocol into a symbolic protocol in the Dolev-Yao model where we can then subject it to automated analysis techniques to prove some symbolic criterion about the

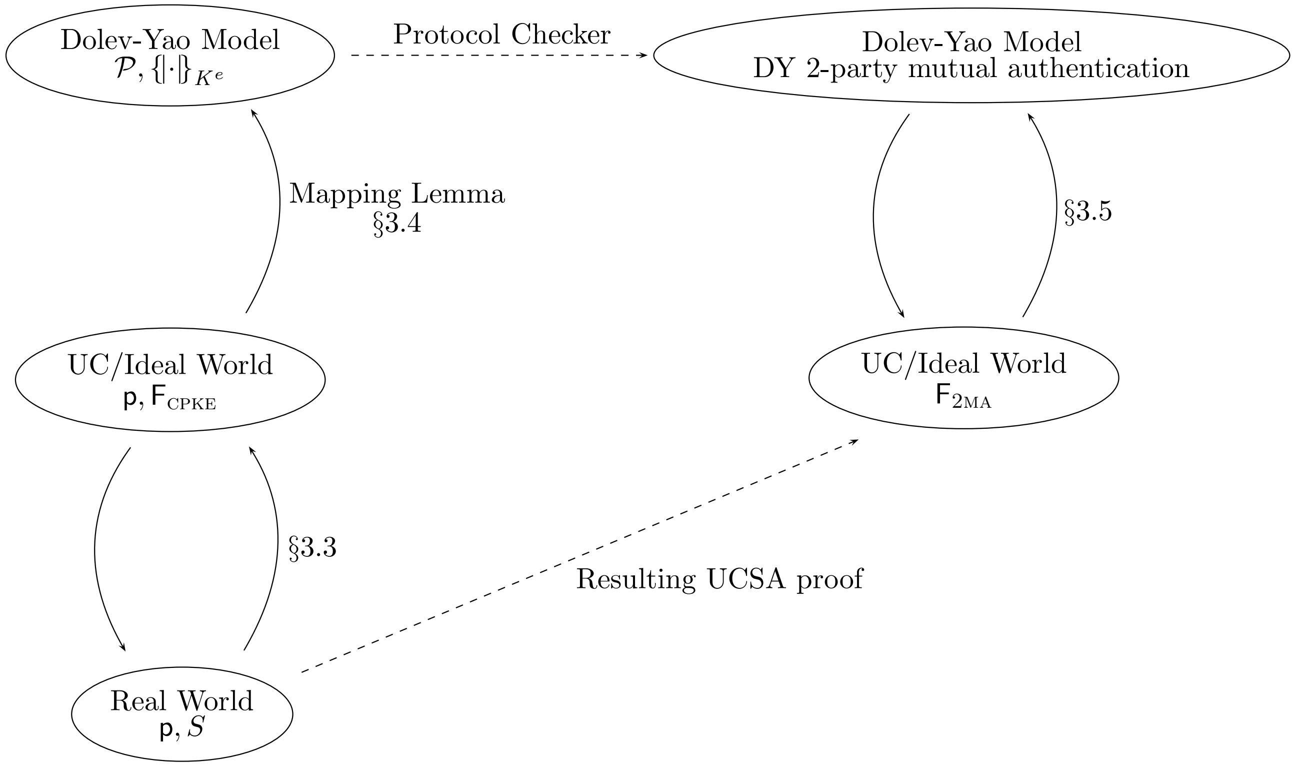

Figure 3-1: A graphical representation of the UCSA framework

protocol. We then step down through the abstraction levels and show that if the symbolic protocol meets a symbolic criterion then the real world protocol achieves a real goal.

The protocol is first transformed into a protocol in the F_{CPKE}-hybrid model, where the use of encryption is replaced by use of ideal encryption functionality F_{CPKE}. Actually, F_{CPKE} represents the ideal public key encryption functionality (F_{PKE}) combined with an idealized key binding functionality since key distribution is implicit in the Dolev-Yao model (i.e. it's just accepted that everyone knows everyone else's public keys). By translating our protocol into the F_{CPKE}-hybrid model, the framework avoids these key distribution issues.

The F_{CPKE}-hybrid version of the protocol is then translated into a symbolic protocol in the Dolev-Yao model. In this form, automated tools can analyze the possible symbolic traces for attacks. This analysis is particularly efficient in the framework since tools only need to analyze the correctness of a single protocol instance, thanks to the guarantees of the UC framework [13, 19]

After proving that the symbolic protocol satisfies a particular symbolic property, it remains to show that the symbolic property implies a real protocol goal. This is done by showing that an ideal functionality that captures the intuition of the concrete protocol's goal is realized by any F_{CPKE}-hybrid protocol whose DY translation satisfies the symbolic property. It then follows that the real-world protocol realizes its goal, as described by the goal's idealized functionality description.

3.2 Simple Protocols

As one might imagine, not all protocols are representable in the Dolev-Yao model. In particular, the translation of a real protocol to a symbolic protocol can only occur if the original protocol is a "simple protocol." A simple protocol is one where participants are only asked to perform actions that are the natural analogues of symbolic operation: concatenation, splitting, nonce generation, encryption, decryption, etc. A protocol that requires an honest party to xor two values together, for example, *would not* be a simple protocol. The Needham-Schroeder protocol, before and after Lowe's fix, *is* a simple protocol. This is a necessary restriction for the universally composable symbolic analysis framework as currently formulated. While this excludes a number of important protocols, the set of possible simple protocols is still a rich one. We formally define simple protocols as follows:

Definition 13 (Simple protocols) *A* simple protocol *is a pair of interactive Turing machines (ITMs)* $\{\mathtt{M}_1, \mathtt{M}_2\}$*, one for each role, where each machine* $\mathtt{M}_i$ *implements an algorithm described by a pair* (Σ, Π)*:*

- Σ *is a* store*, a mapping from variables to tagged values (explained further below) and*
- Π *is a program that expects as input*
  - *The security parameter* k*,*
  - *Its SID* SID*, its PID* PID*, and its RID* RID*,*
  - PID_1 *which represents the name for the other participant of this protocol execution.*

The program tags the input values, binds them to variables in the store, and then acts according to a sequence of commands consistent with the grammar in Figure 3-2.

$$
\begin{array}{rcl}
\Pi & ::= & \text{BEGIN; STATEMENTLIST} \\
\text{BEGIN} & ::= & \texttt{input}(\mathsf{SID}, \mathsf{RID}, \mathsf{PID}_0, \mathsf{PID}_1, \mathsf{RID}_2, ...); \\
 & & \quad \text{(Store } \langle\text{“role”}, \mathsf{RID}\rangle, \langle\text{“name”}, \mathsf{PID}_0\rangle, \langle\text{“name”}, \mathsf{PID}_1\rangle, \langle\text{“name”}, \mathsf{PID}_2\rangle, ... \\
 & & \quad \text{in local variables } \mathsf{MyRole}, \mathsf{MyName}, \mathsf{PeerName}, \mathsf{OtherName}_2, ... \\
 & & \quad \text{respectively.} \\
\text{STATEMENTLIST} & ::= & \text{STATEMENT STATEMENTLIST} \\
 & | & \text{FINISH} \\
\text{STATEMENT} & ::= & \texttt{newrandom}(\mathsf{v}) \\
 & & \quad \text{(generate a } k\text{-bit random string } r \text{ and store } \langle\text{“random”}, r\rangle \text{ in } \mathsf{v}) \\
 & | & \texttt{encrypt}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3}) \\
 & & \quad \text{(Send } (\texttt{Encrypt}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \mathsf{v2}) \text{ to } \mathsf{F}_{\text{CPKE}} \text{ where } v1 = \langle\text{“pid”}, \mathsf{PID}\rangle, \\
 & & \quad \text{receive } c\text{, and store } \langle\text{“ciphertext”}, c, \langle \mathsf{PID}_1, \mathsf{SID}\rangle\rangle \text{ in } \mathsf{v3}) \\
 & | & \texttt{decrypt}(\mathsf{v1}, \mathsf{v2}) \\
 & & \quad \text{(If the value of } \mathsf{v1} \text{ is } \langle\text{“ciphertext”}, c'\rangle \text{ then send} \\
 & & \quad (\texttt{Decrypt}, \langle \mathsf{PID}_0, \mathsf{SID}\rangle, c') \text{ to } \mathsf{F}_{\text{CPKE}} \text{ instance } \langle \mathsf{PID}_0, \mathsf{SID}\rangle, \\
 & & \quad \text{receive some value } m\text{, and store } m \text{ in } \mathsf{v2} \text{ Otherwise, end.} \\
 & | & \texttt{receive}(\mathsf{v}) \\
 & & \quad \text{(Receive message, store in } \mathsf{v}) \\
 & | & \texttt{output}(\mathsf{v}) \\
 & & \quad \text{(send value of } \mathsf{v} \text{ to local output)} \\
 & | & \texttt{pair}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3}) \\
 & & \quad \text{(Store } \langle\text{“pair”}, \sigma_1, \sigma_2\rangle \text{ in } v3\text{, where } \sigma_1 \text{ and } \sigma_2 \text{ are the values of} \\
 & & \quad \mathsf{v1} \text{ and } \mathsf{v2}\text{, respectively.)} \\
 & | & \texttt{separate}(\mathsf{v1}, \mathsf{v2}, \mathsf{v3}) \\
 & & \quad \text{(if the value of } v1 \text{ is } \langle\text{“join”}, \sigma_1, \sigma_2\rangle\text{, store } \sigma_1 \text{ in } v2 \\
 & & \quad \text{and } \sigma_2 \text{ in } v3 \text{ (else end))} \\
 & | & \texttt{if } (\mathsf{v1} == \mathsf{v2} \texttt{ then } \text{STATEMENTLIST} \texttt{ else } \text{STATEMENTLIST} \\
 & & \quad \text{(where } \mathsf{v1} \text{ and } \mathsf{v2} \text{ are compared by value, not reference)} \\
\text{FINISH} & ::= & \texttt{output}(\langle\text{“finished”}, \mathsf{v}\rangle); \texttt{ end}.
\end{array}
$$

The symbols v, $\mathsf{v1}$, $\mathsf{v2}$ and $\mathsf{v3}$ represent program variables. It is assumed that $\langle\text{“pair”}, \sigma_1, \sigma_2\rangle$ encodes the bit-strings σ_1 and σ_2 in such a way that they can be uniquely and efficiently recovered. A party's input includes its own PID, the PID of its peer, and other PIDs in the system. Recall that the SID of an instance of F_{CPKE} is an encoding $\langle \mathsf{PID}, \mathsf{SID}\rangle$ of the PID and SID of the legitimate recipient.

Figure 3-2: The grammar of simple protocols

The structure of simple protocols makes it simple to find the Dolev-Yao counterpart to a simple protocol:

Definition 14 (Mapping of simple protocols to symbolic protocols) *Let* $\mathsf{p} = \{\mathtt{M}_0, \mathtt{M}_1\}$ *be a simple protocol. Then* $\widehat{\mathsf{p}}$ *is the Dolev-Yao protocol*

$$\mathcal{P}_i : \mathcal{S} \times \mathcal{M} \times \mathcal{O} \times \mathcal{A} \to \mathcal{S} \times \mathcal{A} \times \mathit{message} \times \mathcal{A} \times \mathit{output}$$

that implements ITM $\mathtt{M}$*, except that:*

- *The variables of* $\mathtt{M}$ *are interpreted as elements of the symbolic message algebra* $\mathcal{A}$*.*
- *Instead of receiving as input* SID, PID_0, PID_1, RID*, the store is initialized with its own name* P_0*, its own key* K_{P_0}*, and a name* P_1 *and public key* K_{P_1} *of the other participant. The symbols* P_0 *and* P_1 *represent* PID_0 *and* PID_1*, respectively. Similarly, the symbols* K_0 *and* K_1 *represent* $\langle \mathsf{PID}_0, \mathsf{SID} \rangle$ *and* $\langle \mathsf{PID}_1, \mathsf{SID} \rangle$*, respectively.*
- *Instead of creating a new random bit-string, the symbolic protocol returns* $R_{(i,n)}$ *and increments* n *(which starts at 0),*
- *Instead of sending* $(\mathtt{Encrypt}, \langle \mathsf{PID}, \mathsf{SID} \rangle, \mathsf{M})$ *to* F_{CPKE} *and storing the result, the composed symbol* $\{\!|\Sigma(\mathsf{M})|\!\}_{K_{P_1}}$ *is stored instead (where* $\Sigma(\mathsf{M})$ *is the value bound to the variable* M *in the store* Σ*).*
- *Instead of sending* $(\mathtt{Decrypt}, \langle \mathsf{PID}_0, \mathsf{SID} \rangle, \mathsf{C})$ *to* F_{CPKE} *and storing the result, the value stored depends on the form of* $\Sigma(\mathsf{C})$*. If* $\Sigma(\mathsf{C})$ *is of the form* $\{\!|M|\!\}_{K_{P_0}}$ *then the value* M *is stored. Otherwise, the garbage value* $\mathcal{G}$ *is stored instead.*
- *Pairing and separation use the symbolic pairing operator.*
- *Lastly, the bit-strings "starting" and "finished" are mapped to the Dolev-Yao symbols* $\mathit{Starting}$ *and* $\mathit{Finished}$*, respectively.*

3.3 Concrete to UC: F_{PKE} and its Realization

In order to translate a real-world protocol into a F-hybrid protocol, we need to formulate the ideal functionality used by the protocol then determine what computational security definition realizes it.

Proposals for ideal functionality formulations were part of the original description of the UC framework [13], but, over time, a number of changes and corrections have been made to these formulations in conjunction with an increased understanding of the security needs for schemes realizing these functionalities [18, 6, 16, 15].

Canetti and Herzog present a revised formulation of F_{CPKE} (Figure 3-4) which implies the formulation of F_{PKE} given in Figure 3-3.

Functionality F_{PKE}

F_{PKE} proceeds as follows over message domain $\{0,1\}^*$. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{owner}, \mathsf{SID}')$, where PID_{owner} is the identity of a special party, called the owner of this instance.

Key Generation: Upon receiving a value (KeyGen, sid) from some party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, do the following:

1. Hand (KeyGen, sid) to the adversary.
2. Receive a value e from the adversary and hand it to S. If the adversary has not already done so, it also provides the description of deterministic polytime algorithm D.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, e', m)$ from a party P proceed as follows:

1. If $m \notin \mathcal{D}$ then return an error message to P.
2. If $m \in \mathcal{D}$ then
   - If $e = e'$, hand $(\mathtt{Encrypt}, \mathsf{SID}, \mathsf{P})$ to the adversary. Else, hand $(\mathtt{Encrypt}, \mathsf{SID}, e', m, \mathsf{P})$ to the adversary.
   - Receive a tag c from the adversary, record the pair (c, m), and hand c to P. (If c already appears in a previously recorded pair then return an error message to P.)

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand m to P.
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Figure 3-3: The [16] public-key encryption functionality, F_{PKE}

This formulation of F_{PKE} is a variation on the one given by Canetti, Krawczyk, and Nielsen [18] which they proved to be securely realizable by a concrete public key encryption scheme if and only if the scheme is IND-CCA2. As we shall in Chapter 4, this formulation of F_{PKE} is also somewhat flawed, particularly for use in the framework being constructed. This flaw will provide motivation for the redefining of the functionality in Chapter 5.

Functionality F_{CPKE}

F_{CPKE} proceeds as follows, when parameterized by message domain $\mathcal{D}$. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{owner}, \mathsf{SID}')$, where PID_{owner} is the identity of a special party, called the owner of this instance.

Initialization: Expect the first message received from the adversary to contain a description of a deterministic polytime algorithm, D.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, m)$ from a party P proceed as follows:

1. If $m \notin \mathcal{D}$ then return an error message to P.
2. If $m \in \mathcal{D}$ then
   - Hand $(\mathtt{Encrypt}, \mathsf{SID}, \mathsf{P})$ to the adversary. (If the owner of this instance of F_{CPKE} is corrupted, then hand also the entire value m to the adversary.)
   - Receive a tag c from the adversary, record the pair (c, m), and hand c to P. (If *ciphertext* already appears in a previously recorded pair then return an error message to P.)

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand m to P.
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Figure 3-4: The [16] certified public-key encryption functionality, F_{CPKE}

3.4 UC to Dolev-Yao: The Mapping Lemma

In order to be sure protocol execution and adversarial actions are the same in both the UC and Dolev-Yao worlds, we need a way of describing the messages exchanged as well as the meaningful internal states of participants. As you'll recall, in Section 2.3.2 we described the outcome of a symbolic protocol execution with the symbolic trace. We now define an analogue: the concrete protocol trace. Note that per our discussion in Section 3.1, this definition uses ideal functionality F_{CPKE} rather than F_{PKE}.

Definition 15 (Traces of concrete protocols) *Let* p *be a* F*-hybrid protocol. Inductively define* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$*, as the trace of protocol* p *in conjunction with adversary* A *and environment* Z *with inputs* $z, \vec{r}$*, and security parameter* k*. Initially, the trace is the null string. The trace then grows as the protocol's execution progresses.*

- *If the environment provides input* m *to a party with id* $(\mathsf{SID}, \mathsf{RID})$*, then* $\langle \text{“input”}, (\mathsf{SID}, \mathsf{RID}), m\rangle$ *is appended to the end of* t.
- *If the adversary provides input* m *to a party with id* PID*, then* $\langle \text{“adv”}, \mathsf{PID}, m\rangle$ *is*

appended to the end of t*.*

- *If a party* PID *generates a new random string* r*, then* $\langle$ *"random"*$, r\rangle$ *is appended to* t*.*

- *If a party pairs values* m_1 *and* m_2 *to form* (m_1, m_2)*, then* $\langle$ *"pair"*$, m_1, m_2\rangle$ *is appended to* t*.*

- *If a party* PID *writes a message* m*, it does so in one of two ways*

  - *if it writes* m *on its local output tape, then* $\langle$ *"output"*$, \mathsf{PID}, m\rangle$ *is appended to* t*.*

  - *if it writes* m *on its outgoing communication tape, then* $\langle$ *"message"*$, \mathsf{PID}, m\rangle$ *is appended to* t*.*

- *If* F_{CPKE} *is activated by party* PID *with call* $(\mathsf{Encrypt}, \langle\mathsf{PID}, \mathsf{SID}\rangle, m)$ *and* F_{CPKE} *responds with ciphertext* c*, then* $\langle$ *"ciphertext"*$, \langle\mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *is appended to* t*. (If* F_{CPKE} *returns,* $\perp$ *then nothing is appended to* t*).*

- *If* F_{CPKE} *is activated by party* PID *with call* $(\mathsf{Decrypt}, \langle\mathsf{PID}, \mathsf{SID}\rangle, c)$ *and* F_{CPKE} *responds with plaintext* m*, then* $\langle$ *"dec"*$, \langle\mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$ *is appended to* t*. (If* F_{CPKE} *returns,* $\perp$ *then nothing is appended to* t*).*

$\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ *denotes* t *upon completion of the protocol execution. Let* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)$ *denote the random variable for* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ *when* $\vec{r}$ *is uniformly chosen. Let* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}$ *denote the probability ensemble* $\{\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)\}_{k\in\mathbb{N}, z\in\{0,1\}^*}$

Because of the linear manner in which the environment, adversary, and parties are activated in the UC framework, the order in which new elements are appended to the trace is unambiguous for a particular $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$.

We now define a mapping from concrete traces to symbolic traces. It should be clear that the mapping will result in a sequence of interactions between parties and the adversary which resembles a Dolev-Yao trace. The purpose of the subsequent Mapping Lemma is to prove that a concrete protocol's trace maps to a valid Dolev-Yao trace of the corresponding symbolic protocol with $1 - neg(k)$ probability.

Definition 16 (The mapping from concrete traces to symbolic traces) *Let* p *be a concrete* $\mathsf{F}_{\text{CPKE}}/\mathsf{F}_{\text{SIG}}$*-hybrid protocol and let* t *be a trace of an execution of* p *with security parameter* k*, environment* Z *with input* z*, and random input vector* $\vec{r}$*. We determine the* *mapping of* $\mathtt{t}$ *to a Dolev-Yao trace in two steps. (These steps can be thought of as two "passes" on the string* $\mathtt{t}$*.)*

(I.) *First, we read through the string* $\mathtt{t}$ *character by character, in order, and inductively define the following partial mapping* f *from* $\{0,1\}^*$ *to elements of the algebra* $\mathcal{A}$*. (Note that the patterns in* $\mathtt{t}$ *addressed below may be nested and overlapping. That is, the same substring may be part of multiple patterns. A pattern is recognized as soon as the last character in the pattern is read.)*

- *Whenever we encounter a pattern of the form* $\langle \text{"name"}, \beta\rangle$ *for some string* β *and* $f(\langle \text{"name"}, \beta\rangle)$*is not yet defined then set* $f(\langle \text{"name"}, \beta\rangle) = P$ *for some new symbol* $P \in \mathcal{M}$ *not in the range of* f *so far.*

- *Whenever we encounter in some event a pattern of the form* $\langle \text{"random"}, \beta\rangle$ *for some string* β *and* $f(\langle \text{"random"}, \beta\rangle)$ *is not yet defined then set* $f(\langle \text{"random"}, \beta\rangle) = N$ *for some new symbol* $N \in \mathcal{R}$ *that is not in the range of* f *so far.*

- *Whenever we encounter a pattern of the form* $\langle\langle \text{"pid"}, \mathsf{PID}\rangle, \langle \text{"sid"}, \mathsf{SID}\rangle\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}$*, and* $f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *is not yet defined, then set* $f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle) = K$ *for some new* $K \in \mathcal{K}_{Pub}$ *not in the range of* f*.*

- *Whenever we encounter a pattern of the form* $\langle \text{"pair"}, \beta_1, \beta_2\rangle$*, then proceed as follows. First, if* $f(\beta_1)$ *is not yet defined then set* $f(\beta_1) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Similarly, if* $f(\beta_2)$ *is not yet defined then set* $f(\beta_2) = \mathcal{G}$*. Finally, set* $f(\langle \text{"pair"}, \beta_1, \beta_2\rangle) = f(\beta_1)|f(\beta 2)$*.*

- *Whenever we encounter a pattern of the form* $\langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}, m, c$*, then* f *is expanded so that* $f(\langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever an encryption call to* F_{CPKE} *is made. Also, at this point both* $f(m)$ *and* $f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *must already be defined, since this is an encryption call made by a party running a simple protocol.)*

- *Whenever we encounter a pattern of the form* $\langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then proceed as follows. First, if* $f(m)$ *is not yet defined, then set* $f(m) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Next, set* $f(\langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle \text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever a decryption call to* F_{CPKE} *is made. The* *case where $f(m) = \mathcal{G}$ occurs when a ciphertext was not generated via the encryption algorithm. It includes both the case where the decryption algorithm fails and the case where the decryption algorithm outputs a message that cannot be parsed by simple protocols.)*

(II.) *In the second step, we construct the actual Dolev-Yao trace. Let* $\mathsf{t} = \mathsf{G}_1||\mathsf{G}_2||\ldots\mathsf{t}_n$ *be the concrete trace. Then construct the Dolev-Yao trace* $\widehat{\mathsf{t}}$ *by processing each* G *in turn, as follows:*

- *If* $\mathsf{G}_i = \langle\, \text{“input”}, (\mathsf{SID}, \mathsf{RID}), m\rangle$*, then we find* $\mathsf{m} = f(m)$*, and generate the symbolic event* $H = [\,\text{“input”}, P, \mathsf{m}]$ *(where P is the symbolic name of the input recipient).*

- *If* $\mathsf{G}_i = \langle\, \text{“ciphertext”}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *or* $\mathsf{G}_i = \langle\, \text{“dec”}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then no symbolic event is generated.*

- *If* $\mathsf{G}_i = \langle\, \text{“output”}, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle\, \text{“name”}, \mathsf{PID}\rangle), output, f(m)).$$

- *If* $\mathsf{G}_i = \langle\, \text{“message”}, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle\, \text{“name”}, \mathsf{PID}\rangle), message, f(m)).$$

- *If* $\mathsf{G} = \langle\, \text{“adv”}, \mathsf{PID}, m\rangle$*, let* $\mathsf{m} = f(m)$*. Then there are two cases:*

  1. m *is in the closure of the symbolic interpretations of the messages sent by the parties in the execution so far, i.e.*

  $$\mathsf{m} \in C\left[\{\mathsf{m}' : \mathsf{m}' = f(m') \text{ and the event } \langle\, \text{“message”}, \mathsf{PID}, m'\rangle \text{ is a prior event in } t\}\right].$$

  In this case there exists a finite sequence of adversary events that produces m_i*. Then* G *is mapped to this sequence of events* H_{i_1}, $H_{i,2}\ldots H_{i,n'}$ *so that the message of* $H_{i,n'-1}$ *is* m_i *and* $H_{i,n'} = [\,\text{“deliver”}, (i, n'-1), P']$ *(where P' is the Dolev-Yao name of the concrete participant who received the message from the concrete adversary).*

2. *Otherwise,* m *is not in the above closure. In this case,* G *maps to the Dolev-Yao event* $[\text{“}fail\text{”}, \mathsf{m}_i]$.

The desired result is then to show that if t is the concrete trace of a simple protocol p, then the mapping of t, $\widehat{\mathsf{t}}$ is a valid Dolev Yao trace of the symbolic protocol $\widehat{\mathsf{p}}$. This is the Mapping Lemma of [16].

Lemma 2 (The Mapping Lemma) *Let* F *denote the ideal functionality* F_{CPKE}*. For all simple protocols* p*, adversaries* A*, environments* Z*, and inputs* z *of length polynomial in the security parameter* k*,*

$$\Pr\left[\mathsf{t} \leftarrow \text{TRACE}^{\mathsf{F}}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z) : \widehat{\mathsf{t}} \textit{ is a valid DY trace for } \widehat{\mathsf{p}}\right] \geq 1 - neg(k)$$

We'll postpone the Mapping Lemma's proof until Chapter 7 when we will prove the Mapping Lemma for both public key encryption and digital signatures based on the ideal function formulations presented in Chapter 5.

3.5 Dolev-Yao Back to UC: DY Mutual Authentication and F_{2MA}

After a concrete simple protocol has been transformed into a symbolic protocol, we can determine whether or not the symbolic protocol satisfies a certain criterion. Having done this, it remains to be shown that the criterion satisfaction implies the meeting of the concrete protocol's goals. In their paper, Canetti and Herzog consider two different protocol objectives: mutual authentication and key exchange. We limit our discussion in this thesis to mutual authentication. They define Dolev-Yao mutual authentication as follows:

Definition 17 (Dolev-Yao 2-party mutual authentication) *A Dolev-Yao protocol* $\mathcal{P}$ *provides* Dolev-Yao mutual authentication (DY-MA) *if all Dolev-Yao traces for* $\mathcal{P}$ *that include an output message* $\langle Finished|P_0|P_1|\mathsf{m}\rangle$ *by participant* P_0*, where* $P_0, P_1 \notin \mathcal{M}_{Adv}$*, include also a previous input message* $\langle Starting|P_1|P_0|\mathsf{m}'\rangle$ *by* P_1*.*

Or, in other words, the party P_0 only thinks the protocol has completed successfully if the party it wants to authenticate with, P_1, believes it is engaged in the protocol instance with P_0.

Functionality F_{2MA}

1. The functionality F_{2MA} begins with a variable $\mathsf{Finished}$ set to false.
2. Upon receiving an input $(\texttt{Authenticate}, \mathsf{SID}, \mathsf{P}, \mathsf{P}', \mathsf{RID})$ from some party P, where $\mathsf{RID} \in \{Initiator, Responder\}$, do:
   (a) If this is the first input (i.e., no tuple is recorded) then denote $\mathsf{P}_0 = \mathsf{P}$, $\mathsf{P}_1 = \mathsf{P}'$, and record the pair $(\mathsf{P}_0, \mathsf{P}_1)$.
   (b) Else, if the recorder pair $(\mathsf{P}_0, \mathsf{P}_1)$ satisfies $\mathsf{P} = \mathsf{P}_1$ and $\mathsf{P}' = \mathsf{P}_0$, set $\mathsf{Finished}$ to true.
   (c) In either case, send the pair $(\mathsf{P}, \mathsf{P}'), \mathsf{RID}$ to the simulator.
3. Upon receiving from the simulator a request $(\texttt{Output}, \mathsf{SID}, X)$, if X is either P_0 or P_1, and $\mathsf{Finished}$ is true then send $\texttt{Finished}$ to X. Else, do nothing.

Figure 3-5: The 2-party mutual authentication functionality

They then showed that for ideal 2-party mutual authentication (Figure 3-5), the following holds:

Theorem 3 *Let* p *be a simple two-party protocol. Then* p *realizes* F_{2MA} *if and only if the corresponding symbolic protocol* $\widehat{\mathsf{p}}$ *satisfies Dolev-Yao 2-party mutual authentication*

At the end of this chain of protocol transforming, mapping, and constraint proving, we finally have a computationally sound argument that a concrete protocol realizes its protocol goal as described by an ideal functionality. Using this framework, Canetti and Herzog describe the process for proving that for protocol p instantiated with IND-CCA2 secure encryption and a sufficiently secure key distribution system, if $\widehat{\mathsf{p}}$ satisfies Dolev-Yao 2-party mutual authentication, then p realizes F_{2MA}. Or, in the language of UC traces,

$$\text{IDEAL}_{\mathsf{F}_{\text{2MA}},\mathsf{S},\mathsf{Z}} \approx \text{EXEC}_{\mathsf{p},\mathsf{A},\mathsf{Z}}.$$

Chapter 4

Analyzing Previous Ideal Functionality Formulations

Ideal functionalities capture our intuition for how a cryptographic operations should occur. Armed with an ideal functionality description, we can find concrete schemes and protocols that securely realize the ideal and use them in protocols secure in the knowledge that they will behave in a manner we'd expect. Correctly defining an ideal functionality, however, has proven to be more difficult than previously thought, as evidenced by the constant revision of proposed ideal functionalities for seemingly straightforward cryptographic concepts [12, 18, 16, 17, 6]. In this section we draw attention to an implicit assumption made in many functionality formulations which leads to undesired behavior, particularly when the functionality is used in the UCSA framework. We then analyze the current formulations of ideal public key encryption and digital signature functionalities, identifying properties that are present but undesired or desired by not present in these formulations.

In particular, the formulation for ideal public key encryption used in [16] (F_{PKE}, Figure 3-3) provides too much power to the adversary, allowing the adversary to learn information it should not have access to. This over-empowerment was missed when constructing the UCSA framework and, uncorrected, invalidates the framework's Mapping Lemma.

The current formulation for ideal signatures (F_{SIG} [15], Figure 4-3) is valid in the environment for which it was proposed – one where it is assumed that all signatures generated are public knowledge. However, when the signature functionality is combined with encryption functionality, this is not longer true and a different formulation must be defined if signatures are to be used in conjunction with other cryptographic primitives in protocol proving frameworks such as the UCSA framework.

Throughout the chapter, we will illustrate these undesired ideal functionality behaviors using example protocols. In these examples we employ the following notation for readability and to avoid ambiguity with Dolev-Yao notation. For public key encryption, we use $enc_A(m)$ to denote the ciphertext for message m, encrypted under party A's encryption key. For digital signatures we use $sig_A(m)$ to denote the signature of message m for party A's verification key.

4.1 Implicit Restrictions on Ideal Realizations

The ideal functionality literature has primarily focused on realizing ideal functionalities through use of stateless (or "memoryless"), local algorithms. By "stateless," we mean each algorithm does not maintain state between invocations; by "local," we mean there is no direct communication between the invocations of a scheme's algorithms (e.g. my instance of *enc* does not directly communicate with your instance of *dec* or even with my own instance of *dec*). This focus is often made implicitly and shortchanges the power of the UC framework – the ability to fully abstract the implementation details of a functionality's underlying implementation.

If a proof showing a particular security type realizes an ideal functionality assumes the implementing scheme lacks state and is local, then the results of the proof may not extend to schemes that do not meet this assumption. For example, a signature scheme where *sig* appends all previous signatures to each new signature is still technically EUF-ACMA, but violates our intuition for how a secure signature scheme should act.[1] There has been work on stateful signature schemes [7, 15], but such papers are less common and, at times, miss some subtleties of this condition.[2]

Taking these schemes into consideration is important. In addition to schemes that purposefully maintain state or use remote input, many schemes which *are* local and stateless are used in such a way that introduces state or external input. A real world use of an

[1]To see why, consider a protocol where a party A takes a message m, and broadcasts $enc_B(m, sig_A(m))$. An adversary (who is not party B) should remain unable to produce A's signature on message m. However, the moment party A signs some other message m' and communicates $(m', sig_A(m'))$ in the clear, the adversary learns the previous signature $sig(m)_A$ despite our intuition that it should remain hidden.

[2]Canetti's [15] formulation of $\mathsf{F}_{\mathrm{SIG}}$ is, in fact, realizable by EUF-ACMA schemes with state. The problem, as we shall see in Section 4.3.2, stems from the $\mathsf{F}_{\mathrm{SIG}}$ formulation itself.

encryption or signature scheme might reset its key after some fixed number of uses. A protocol might call for encrypted messages to contain a counter that increases with each message exchanged. A device that holds secret plaintexts may encrypt any message but only decrypt ciphertexts it generates itself. There is a uniform and intuitive sense of the security/behavior we expect from any encryption or signature scheme, and definitions such as IND-CCA2 and EUF-ACMA only capture it for local, stateless algorithms. A properly formulated description of an ideal functionality, however, can serve as a standard for security expectations, independent of the underlying scheme details.

4.2 Public Key Encryption

In past formulations of F_{PKE} and F_{CPKE} (Figures 3-3 and 3-4), the ideal functionalities have allowed the adversary to choose the ciphertext strings. Since it was still difficult for adversaries to decrypt ciphertexts, it was felt that allowing adversaries to choose these values avoided placing too strong a limitation on concrete schemes realizing the ideal functionality.

Giving the adversary this power means the adversary knows the value of all ciphertexts generated by protocol participants. As the double encryption protocol in figure 4-1 illustrates, this is a problem:

$$\begin{array}{lc} A \rightarrow B: & R \\ A \leftarrow B: & enc_A\{R, enc_B\{m\}\} \end{array}$$

Figure 4-1: A double encryption protocol

For honest participants A and B, it is clear that a passive adversary watching this exchange only learns R and $enc_A\{R, enc_B\{m\}\}$. However, in an environment with ideal functionalities (Figure 4-2), a passive adversary learns the values R, $enc_B\{m\} = c$, and $enc_A\{R, enc_B\{m\}\} = c'$.

The F_{PKE} ideal functionality does not require any unpredictability of the ciphertext, other than message independence. This unpredictability is a desired property, it just happens that its specification is not needed when being realized by stateless encryption schemes. To make this requirement explicit, we will modify F_{PKE} in Section 5.1 to no longer ask the adversary for encryption values. Instead, the adversary will provide a randomized algorithm which

$$
\begin{array}{lll}
A \rightarrow B : R & & \\
& B \rightarrow \mathsf{F}_{\text{CPKE}} : (\mathsf{encrypt}, m) & \\
& & \mathsf{F}_{\text{CPKE}} \rightarrow Adv : (\mathsf{encrypt}, B) \\
& & \mathsf{F}_{\text{CPKE}} \leftarrow Adv : (\mathsf{ciphertext}, c) \\
& B \leftarrow \mathsf{F}_{\text{CPKE}} : (\mathsf{ciphertext}, c) & \\
& B \rightarrow \mathsf{F}_{\text{CPKE}} : (\mathsf{encrypt}, A, (R, c)) & \\
& & \mathsf{F}_{\text{CPKE}} \rightarrow Adv : (\mathsf{encrypt}, A) \\
& & \mathsf{F}_{\text{CPKE}} \leftarrow Adv : (\mathsf{ciphertext}, c') \\
& B \leftarrow \mathsf{F}_{\text{CPKE}} : (\mathsf{ciphertext}, c') & \\
A \leftarrow B : c & &
\end{array}
$$

Figure 4-2: The double encryption protocol, expanded

will be executed to generate values. The adversary's power in this new definition is strictly weaker than in the previous definition.

4.3 Signatures and Certification with Encryption

Literature concerning ideal signature functionality has generally limited itself to an environment where the signature functionality is the only one present. While determining a satisfactory formulation in this type of environment is useful, it is merely a precursor to a more powerful concept of an ideal signature functionality in an environment that includes other ideal functionalities. Such a formulation would allow security proofs for protocols involving multiple cryptographic primitives.

Unfortunately, the limited version of ideal signature functionality does not trivially translate into one suited for coexistence with other ideal functionalities. In this subsection we describe some of the inadequacies with the current F_{SIG} and derivative F_{CERT} formulations.

4.3.1 F_{SIG} and F_{CERT}

In [14, 15], Canetti presented a revised formulation for ideal signature functionality which we reproduce in Figure 4-3 with some slight modifications for consistency. He showed this formulation of F_{SIG} is realizable by a concrete signature scheme if and only if the scheme is EUF-ACMA and could be combined with an ideal certificate authority F_{CA} to produce ideal certification F_{CERT} (Figure 4-4). In these formulations, it is assumed there are no encryption like ideal functionalities in the environment.

Functionality F_{SIG}

Key Generation: Upon receiving a value (`KeyGen`, sid) from some party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, hand (`KeyGen`, sid) to the adversary. Upon receiving (`Verification Key`, sid, v) from the adversary, output (`Verification Key`, sid, v) to S, and record the pair (S, v). v is now the verification key for S.

Signature Generation: Upon receiving a value (`Sign`, sid, m) from S,

1. Verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request.
2. Send (`Sign`, sid, m) to the adversary.
3. Upon receiving (`Signature`, sid, m, σ) from the adversary, verify that no entry $(m, \sigma, v, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output (`Signature`, sid, m, σ) to S, and record the entry $(m, \sigma, v, 1)$.

Signature Verification: Upon receiving a value (`Verify`, sid, m, σ, v') from some party P, hand (`Verify`, sid, m, σ, v') to the adversary. Upon receiving (`Verified`, sid, m, ϕ) from the adversary do:

1. If $v' = v$ and the entry $(m, \sigma, v, 1)$ is recorded, then set $f = 1$.
2. Else, if $v' = v$, the signer is not corrupted, and no entry $(m, \sigma', v, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, v, 0)$.
3. Else, if there is an entry (m, σ, v', f') recorded, then let $f = f'$.
4. Else, let $f = \phi$ and record the entry (m, σ, v', ϕ).

Output (`Verified`, id, m, f) to P.

Figure 4-3: The [14] signature functionality, F_{SIG}.

Functionality F_{CERT}

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from S,

1. Verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request.
2. Send $(\texttt{Sign}, sid, m)$ to the adversary.
3. Upon receiving $(\texttt{Signature}, sid, m, \sigma)$ from the adversary, verify that no entry $(m, \sigma, 0)$ is recorded. If it is, then output an error message to S and halt. Else, output $(\texttt{Signature}, sid, m, \sigma)$ to S, and record the entry $(m, \sigma, 1)$.

Signature Verification: Upon receiving a value $(\texttt{Verify}, sid, m, \sigma)$ from some party P, hand $(\texttt{Verify}, sid, m, \sigma)$ to the adversary. Upon receiving $(\texttt{Verified}, sid, m, \phi)$ from the adversary do:

1. If $(m, \sigma, 1)$ is recorded, then set $f = 1$.
2. Else, if the signer is not corrupted and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.
3. Else, if there is an entry (m, σ, f') recorded, then let $f = f'$.
4. Else, let $f = \phi$ and record the entry (m, σ, ϕ).

Output $(\texttt{Verified}, id, m, f)$ to P.

Figure 4-4: The [14] certification functionality, F_{CERT}.

4.3.2 Adversarial Signature Selection

As with F_{CPKE}, F_{SIG} allows the adversary to select signature values. If signature functionality exists in an environment with encryption, this leads to the same problem demonstrated in Section 4.2. A slight modification (Figure 4-5) of our earlier counterexample illustrates this weakness.

$$
\begin{aligned}
A \rightarrow B: &\quad m \\
A \leftarrow B: &\quad enc_A\{m, sig_B\{m\}\}
\end{aligned}
$$

Figure 4-5: A mixed protocol

Again, a passive adversary should only learn m and $enc_A\{m, sig_B\{m\}\}$, but in an environment with ideal functionalities (Figure 4-6), a passive adversary learns the value for m, $sig_B\{m\} = \sigma$, and $enc_A\{m, sig_B\{m\}\} = c$.

As with public key encryption ideal functionality, this will require the use of a randomized function for generating signature values.

$$
\begin{array}{lll}
A \to B : R & & \\
& B \to \mathsf{F}_{\text{CERT}} : (\mathsf{sign}, m) & \\
& & \mathsf{F}_{\text{CERT}} \to Adv : (\mathsf{sign}, B) \\
& & \mathsf{F}_{\text{CERT}} \leftarrow Adv : (\mathsf{signature}, \sigma) \\
& B \leftarrow \mathsf{F}_{\text{CERT}} : (\mathsf{signature}, \sigma) & \\
& B \to \mathsf{F}_{\text{CPKE}} : (\mathsf{encrypt}, A, (m, \sigma)) & \\
& & \mathsf{F}_{\text{CPKE}} \to Adv : (\mathsf{encrypt}, A) \\
& & \mathsf{F}_{\text{CPKE}} \leftarrow Adv : (\mathsf{ciphertext}, c) \\
& B \leftarrow \mathsf{F}_{\text{CPKE}} : (\mathsf{ciphertext}, c) & \\
A \leftarrow B : c & &
\end{array}
$$

Figure 4-6: The mixed protocol, expanded

4.3.3 Signature Looseness

Many common definitions of signature security allow for public modification of signatures. Intuitively, a signature scheme loses little by allowing an adversary to construct a valid message signature pair (m, σ) if it already knows of a different valid signature σ' on message m. In fact, it seems almost worrisome that a scheme that protects against such adversarial action might be prohibitively restrictive – perhaps forcing signers to only sign a message once.

It becomes difficult, however, for ideal functionalities to allow entities to create alternate signatures if it is unclear to the functionality at the time of verification that the authoring entity had seen a valid signature at the time of creation. In F_{SIG}, if a message has been validly signed in the past, then the adversary is allowed to choose the validity of any proposed alternate signatures. This presents a problem when ideal signature functionality is combined with ideal encryption functionality.

Consider a "vouching" protocol. Party A is taking messages, but only from parties who are currently authorized to do so by some third party C whom A does not have direct communication with. Parties A, B, C might engage in a protocol like that of Figure 4-7 to allow B to send the message to A.

$$
\begin{array}{ll}
A \to B : sig_A\{R\} & \\
& B \to C : sig_A\{R\} \\
& B \leftarrow C : enc_B\{sig_C\{R\}\} \\
A \leftarrow B : enc_A\{m, sig_C\{R\}_C\} &
\end{array}
$$

Figure 4-7: A vouching protocol

When A verifies the final message of the protocol, the adversary has not yet seen C's signature of R. This does not, however, stop the adversary from stepping in at the final stage of the protocol with a forgery of C's signature:

$$A \leftarrow Adv : enc_A\{m', sig_C\{R\}'\}$$

Because a valid signature for R exists under C's key, when A goes to F_{CERT} to verify this forgery, the adversary can choose to have F_{CERT} accept the signature, meaning the adversary has produced C's signature on R without knowing it beforehand.

This problem will require us to examine the looseness of the ideal signature formulation (Section 5.2) and schemes that realize it (Section 6.2).

Chapter 5

Redefining Ideal Functionalities

We redefine the formalizations of ideal functionalities F_{CPKE}, F_{SIG}, F_{CERT} based on the observations of Chapter 4. These new formalizations make use of "statistically unpredictable" algorithms which are algorithms that, intuitively, "look random," even to an adversary that has been able to observe the algorithm's output on adaptively chosen inputs.

To be more precise, *any* adversary, even a computationally unbounded one with adaptive oracle access to G, is unable to guess, with non-negligible probability, G's next output for *any* input. We formally define statistically unpredictable algorithms as follows:

Definition 18 (Statistically Unpredictable) *PPT algorithm G is* statistically unpredictable with respect to k *("statistically unpredictable" for short) if for any* computationally unbounded *adversary A interacting with an instance of G with no initial state,*

$$\Pr[\quad \begin{array}{l} (x, y) \leftarrow A^{G(\cdot,1^k)}(1^k); \\ y' \leftarrow G(x, 1^k) : \\ y = y' \end{array} \quad] < neg(k)$$

Note that the instance of G that our adversary has oracle access to is the same instance that produces output y' (after our adversary has announced its guess y).

A statistically unpredictable algorithm may maintain state as long as it remains unpredictable with each call. We do, however, require that the algorithm be locally executable, i.e. $G(x)$ is executable on a PPT interactive Turing machine requiring no additional inputs other than x. While the adversary is allowed to interact with the same instance of G all it wants, it does not have access to G's randomness.

Statistical unpredictability differs from pseudo-randomness because the algorithm is randomized and is even unpredictable on inputs selected by the adversary. This definition is also different from semantic security (where an adversary cannot tell the difference between $G(m)$ and $G(m')$ for m and m' of its choosing) since we do not care if m is recoverable from $G(m)$, merely that $G(m)$ is hard to predict for any m.

$\mathsf{F}_{\text{PKE}}/\mathsf{F}_{\text{CPKE}}$ and $\mathsf{F}_{\text{SIG}}/\mathsf{F}_{\text{CERT}}$ maintain lists of acceptable encryption (E) or signing (S) algorithms, respectively, that are known to be poly-time and statistically unpredictable. In the initial steps, when the adversary provides the algorithm descriptions, the adversary is only allowed to choose algorithms from this list. This is analogous to an adversary picking from the suite of encryption/signature schemes that are known to have the aforementioned properties.

Note that we do not check, when creating ciphertexts/signatures, if the generated value has either been created before or used by the adversary. By the statistical unpredictability of the relative algorithms, the probability that a collision happens is negligible.

5.1 Public Key Encryption

In order to address the weakness in F_{PKE} described in Section 4.2, we redefine the ideal functionality (Figure 5-1) using the notion of statistical unpredictability. Users of public key encryption want a functionality which allows them to use another party's public key to encrypt a message such that the other party can decipher the message, but anyone else who sees the ciphertext learns nothing about the plaintext.

Informally, we want to simulate the functionality needed by users of the public key encryption while giving the adversary as much power as possible. When a party needs to generate a public key, we let the adversary pick the value for the key. When we need to encrypt a message m, we run a statistically unpredictable algorithm, E, selected by the adversary on $0^{|m|}$. We store the resulting ciphertext c along with plaintext m in our memory and hand back c.[1] If the encrypting party uses the wrong encryption key, then we give m to the adversary and return a ciphertext c of the adversary's choosing (as long as c isn't a repeat). When the owner of the public key wants to decrypt a ciphertext, we check if we created the ciphertext – if so, we return the plaintext we remember for that ciphertext; if

[1] Because E is only given $0^{|m|}$, it is impossible to guess anything about m other than its length. Because E is statistically unpredictable, the adversary is unable to guess the value of c.

not, we run a decryption algorithm D that the adversary gave us when it created the public key.

Functionality F_{PKE}

F_{PKE} proceeds as follows over message domain $\{0,1\}^*$. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{owner}, \mathsf{SID}')$, where PID_{owner} is the identity of a special party, called the owner of this instance.

Key Generation: Upon receiving a value (KeyGen, sid) from some party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, do the following:

1. Hand (KeyGen, sid) to the adversary.
2. Receive a public key value e from the adversary and hand it to S. If the adversary has not already done so, it also provides descriptions of statistically unpredictable polytime algorithm E and deterministic polytime algorithm D.
3. Record the value e.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, e', m)$ from a party P proceed as follows:

1. If $m \notin D_k$ return an error message to P.
2. If $e' = e$, set $c = \mathsf{E}(0^{|m|}, 1^k)$, record pair (c, m), and return c to P.
3. If $e' \neq e$, hand $(\mathsf{Encrypt}, sid, e', m)$ to the adversary and set c to its response. If c already appears in a previously recorded pair, then return an error message to P, otherwise return c to P.

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand m to P.
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Figure 5-1: The public-key encryption functionality, F_{PKE}

Combining F_{PKE} with ideal key registration functionality F_{REG}, we can create ideal certified public key encryption functionality F_{CPKE} [16], as described in Figure 5-2. In F_{CPKE} we don't have to worry about a party using the wrong key to encrypt a message.

5.2 Signatures and Certification with Encryption

A user of a signatures scheme desires the ability to output a verification key such that only the user can produce the signature for a message under the verification key, meaning any message accompanied with a valid signature must have been signed by the user.

As discussed in Section 4.3.3, loose signatures present a problem when formulating an ideal signature functionality F_{SIG} in a model containing encryption functionality. In order to

Functionality F_{CPKE}

F_{CPKE} proceeds as follows over message domain $\{0,1\}^*$. The SID is assumed to consist of a pair $\mathsf{SID} = (\mathsf{PID}_{owner}, \mathsf{SID}')$, where PID_{owner} is the identity of a special party, called the owner of this instance.

Initialization: Expect the adversary to provide the descriptions of statistically unpredictable polytime algorithm E and deterministic polytime algorithm, D.

Encryption: Upon receiving a value $(\mathsf{Encrypt}, \mathsf{SID}, m)$ from a party P proceed as follows:

1. Set $c = \mathsf{E}(0^{|m|})$.
2. Record pair (c, m), and hand c to P.

Decryption: Upon receiving a value $(\mathsf{Decrypt}, \mathsf{SID}, c)$ from the owner of this instance, proceed as follows. (If the input is received from another party then ignore.)

1. If there is a recorded pair (c, m), then hand m to P.
2. Otherwise, compute $m = \mathsf{D}(c)$, and hand m to P.

Figure 5-2: The certified public-key encryption functionality, F_{CPKE}

achieve such a formulation we must redefine F_{SIG} to no longer allow loose signatures (Figure 5-3).

The description of ideal signature functionality is very similar to ideal public key encryption but with some important differences. When asked to sign m, we give m to S since we don't care if signature σ reveals m, we only care that σ is hard for the adversary to guess. For each message/signature pair we see, we remember the verification key it was associated with and whether or not the signature was valid (the last bit b in the four-tuple). We always honor signatures we created and act consistently on any message/signature pair we've seen before. If someone tries to verify a forged message/signature pair under the correct verification key, we reject it; if they're verifying with the wrong key, then we allow the adversarial algorithm V to choose the signature's validity.

The ideal signing functionality F_{SIG} can be combined with an ideal certification authority (F_{CA}, as described in [15]) to create an ideal certification functionality F_{CERT}, described in Figure 5-4. This allows us to remove the case when a party tries to verify with the wrong key.

Functionality F_{SIG}

Key Generation: Upon receiving a value $(\texttt{KeyGen}, sid)$ from some party S, verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request. Else, hand $(\texttt{KeyGen}, sid)$ to the adversary. Upon receiving $(\texttt{VerificationKey}, sid, \mathsf{v}, \mathsf{S}, \mathsf{V})$ from the adversary, output $(\texttt{VerificationKey}, sid, \mathsf{v})$ to S, and record $(S, \mathsf{v}, \mathsf{S}, \mathsf{V})$. S and V are the descriptions of a statistically unpredictable polytime algorithm and a polytime algorithm, respectively. v is a verification key.

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from S,

1. Verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request.
2. Set $\sigma = \mathsf{S}(m)$.
3. Output $(\texttt{Signature}, sid, m, \sigma)$ to S and record the entry $(m, \sigma, 1)$.

Signature Verification: Upon receiving a value $(\texttt{Verify}, \mathsf{SID}, m, \sigma, \mathsf{v}')$ from some party P, proceed as follows.

1. If there is an entry (m, σ, b') recorded, then set $b = b'$.
2. Else, if $\mathsf{v}' = \mathsf{v}$ and the signer is not corrupted, then set $b = 0$ and record the entry $(m, \sigma, 0)$.
3. Else, set $b = \mathsf{V}(m, \sigma, \mathsf{v}')$

Output $(\texttt{Verified}, sid, m, b)$ to P.

Figure 5-3: The signature functionality, F_{SIG}

Functionality F_{CERT}

Initialization: Expect the adversary to provide the descriptions of statistically unpredictable, polytime algorithm S and polytime algorithm, V.

Signature Generation: Upon receiving a value $(\texttt{Sign}, sid, m)$ from S,

1. Verify that $sid = (S, sid')$ for some sid'. If not, then ignore the request.
2. Set $\sigma = \mathsf{S}(m)$.
3. Output $(\texttt{Signature}, sid, m, \sigma)$ to S and record the entry $(m, \sigma, 1)$.

Signature Verification: Upon receiving a value $(\texttt{Verify}, \mathsf{SID}, m, \sigma)$ from some party P, proceed as follows.

1. If there is an entry (m, σ, b') recorded, then set $b = b'$.
2. Else, if the signer is not corrupted, set $b = 0$ and record the entry $(m, \sigma, 0)$.
3. Else, set $b = \mathsf{V}(m, \sigma)$ and record the entry (m, σ, b).

Output $(\texttt{Verified}, sid, m, b)$ to P.

Figure 5-4: The certification functionality, F_{CERT}

Chapter 6

Realizing Ideal Functionalities

In papers concerning ideal functionalities, the relation between a concrete scheme and an ideal functionality is generally described as an equivalence, meaning not only does a particular security definition *realize* the ideal functionality, but the ideal functionality *implies* the security definition is needed. These papers generally assume, however, that the concrete scheme will use local stateless algorithms. Under this assumption, we too will show that our new ideal functionality formulations are in equivalence with existing security definitions. However, as discussed in Section 4.1, these security definitions are not sufficient for proofs involving algorithms that do not meet this criteria

6.1 Public Key Encryption

In our new formulation of F_{PKE} (Figure 5-1) the adversary's powers are strictly weaker than in the previous formulation (Figure 3-3 from [18]). This implies that any local stateless encryption schemes will still need to be at least indistinguishable under a 2-stage chosen ciphertext attack (IND-CCA2) in order to realize F_{PKE}. It remains to be shown, however, that IND-CCA2 security is strong enough to realize this new F_{PKE}.

Definition 19 *Define protocol π_S using encryption scheme S as follows*

1. *When activated, within some P_i and with input (KeyGen, id), run algorithm gen, output the encryption key e and record the decryption key d.*

2. *When activated, within some party P_j and with input $(\mathsf{Encrypt}, id, e', m)$, return $enc(e', m)$.*

3. When activated, within P_i and with input $(\mathsf{Decrypt}, id, c)$, return $dec(d, c)$.

Theorem 4 *Let $S = (gen, enc, dec)$ be an encryption scheme over domain $\mathcal{D}$ composed of local, stateless algorithms. Then π_S securely realizes F_{PKE} with respect to domain $\mathcal{D}$ and non-adaptive adversaries if and only if S is IND-CCA2 secure.*

Proof. **"Only if" -** Assume that π_S securely realizes F_{PKE} but S is not IND-CCA2 secure. This implies that either S is not complete, or there exists an adversary F that can win the CCA game (Figure 2-1) with a non-negligible advantage.

1. Assume Σ is not complete, i.e. there exists $m \in M_k$ such that $\Pr[(e, d) \leftarrow gen(1^k);\ c \leftarrow enc(e, m);\ m' \leftarrow dec(d, c) : m \neq m'] > neg(k)$ for infinitely many k's. In this case, Z sets $sid = (\mathsf{P}, 0)$ and activates some uncorrupted party P with input (KeyGen, sid) to obtain encryption key e. It then activates some other (uncorrupted) party with $(\mathsf{Encrypt}, sid, m)$ to produce c. Z reactivates P with $(\mathsf{Decrypt}, sid, c)$ to obtain m'. and outputs whether $m = m'$

   When interacting with the ideal process, Z clearly always outputs 1, whereas in the interaction with π_S, Z will output 0 with non-negligible property. Thus, a π_S that realizes F_{PKE} must use a S that exhibits completeness.

2. Assume there exists an adversary F for S who wins the CCA game with a non-negligible advantage. Using F, we will construct an environment Z that can distinguish between interactions in the ideal process and in real-life.

   When interacting with a network with two uncorrupted parties P_1 and P_2, Z simulates a copy of F and does the following:

   (a) Z activates P_1 with input (KeyGen, sid) for some random sid. It takes the public key e that is output by P_1 and hands it to F.

   (b) When F makes a decryption query c, Z activates P_1 with input $(\mathsf{Decrypt}, sid, c)$ and hands the resulting plaintext m back to F.

   (c) When F outputs its two plaintext selections m_0 and m_1, Z chooses a bit b at random and activates P_1 with input $(\mathsf{Encrypt}, sid, e, m_b)$. It then hands the resulting challenge ciphertext c^* to F.

(d) When F makes a subsequent decryption query c, Z checks if $c = c^*$. If it does, Z outputs a random bit and halts. As long as $c \neq c^*$, Z activates P_1 with input $(\mathsf{Decrypt}, sid, c)$ and hands the resulting plaintext m back to F.

(e) When F outputs a guess b', Z outputs $b \oplus b'$ and halts.

When Z is operating in the real-life model with adversary A and π_S, its simulated F sees a IND-CCA2 game using encryption scheme S. This implies that F will output $b' = b$ with probability $\frac{1}{2} + \epsilon$ where ϵ is a non-negligible advantage, causing Z to output 0 with probability $\geq \frac{1}{2} + \epsilon$.

When Z is operating in the ideal process with any ideal adversary S and F_{PKE}, then the challenge ciphertext returned to F is generated independent of b. The ciphertext c^* is actually the output of $\mathsf{E}(0^{|m_b|})$ where $|m_0| = |m_1|$. This means F is playing a guessing game where no advantage is possible and will guess $b' = b$ with probability exactly $\frac{1}{2}$.

Thus Z outputs 0 with non-negligibly higher probability when interacting with the real world, violating our initial assumption. Thus there must not exist an adversary F who can regularly win the CCA game under S.

This implies that if π_S securely realizes F_{PKE}, then S is IND-CCA2 secure.

"If" - Assume S is a local, stateless IND-CCA2 secure encryption scheme but that π_S does not realize F_{PKE}, i.e. there is a real-life adversary A such that for any ideal-process adversary S there exists an environment Z that can distinguish whether it is interacting with A and π_S or S and F_{PKE}. Since Z succeeds for any S, it must succeed for the following S:

- S runs a simulated copy of A, denoted A_S.
- Any input from Z is forwarded to A_S. A_S's outputs are copied to S's outputs.
- When S receives request (KeyGen, sid) from F_{PKE}, S runs $gen(1^k)$ to obtain public encryption key e and private decryption key d. It then returns e as well as $\mathsf{E} = enc(e, \cdot)$ and $\mathsf{D} = dec(d, \cdot)$ to F_{PKE}.

Assume there is an environment Z that can distinguish between interactions. Without loss of generality, we assume that in each execution, there are n parties asked to create encryption keys and Z asks for each key to encrypt exactly p messages. We use Z to construct the following algorithm G:

1. G receives a public key e and chooses numbers $j \overset{\mathcal{R}}{\leftarrow} \{1, ..., n\}$ and $h \overset{\mathcal{R}}{\leftarrow} \{1, ..., p\}$.

2. G simulates a copy of Z and simulates its interaction with a system running π_S and A.

3. When Z activates party P_j with input (KeyGen, sid), G has P_j output the value e from G's input. (All other parties run gen when asked to generate a key).

4. At first, when Z instructs a party to encrypt a message $m_{(l,i)}$ under e_{P_l} (where e_{P_l} is P_l's encryption key), the encrypter outputs $c_{(l,i)} = enc(e_{P_l}, m_{(l,i)})$. (For $l = j$, output $enc(e, m_{(j,i)})$).

5. At the h-th request to encrypt a message under e $(m_{(j,h)})$, G outputs $(m_0, m_{(j,h)})$ to its challenger for the IND-CCA2 game, where m_0 is a fixed message in $\mathcal{D}$. Upon receiving back challenge ciphertext c^*, it has the encrypter output $c_{(j,h)} = c^*$ as the encryption of $m_{(j,h)}$.

6. Subsequently, when Z instructs a party to encrypt a message m to under e_{P_l}, G has the encrypter output $enc(e_{P_l}, m_0)$, where m_0 is the same fixed message as before. (For $l = j$, output $enc(e, m_0)$).

7. Whenever a party P_l is activated with input $(\mathsf{Decrypt}, sid, c)$ where $c = c_{(l,i)}$ for some i, G lets P_l output plaintext $m_{(l,i)}$ (even for those cases where the $c_{(l,i)} = enc(e_{P_l}, m_0)$). If $P_l \neq P_j$ and $c \neq c_{(l,i)}$ for all i, then P_l responds with $dec(d_{P_l}, c)$. If $l = j$ but $c \neq c_{(j,i)}$ for all i, then G sends c to the decryption oracle and has P_j output the returned plaintext m.

8. When Z outputs bit b' and halts, G outputs b' and halts as well.

We now argue that G succeeds at guessing b with non-negligible probability using a hybrid argument. Let $\mathsf{Z}(\phi)$ denote the output of Z after observing a simulated execution in which case, Z sees ciphertexts $c_1, ..., c_{\phi-1}, c_\phi, c_{\phi+1}, ...c_{np-1}, c_{np}$ where c_i for $i \geq \phi$ is the encryption of fixed message m_0 under the appropriate encryption key. Note that when all simulated parties (Z+P_l's) have the same randomness, the messages requested for encryption, up to the ϕth message, will be the same for $\mathsf{Z}(\phi)$ and $\mathsf{Z}(\phi+1)$, but subsequent encryption requests may not be the same.

By our original assumption, Z is able to distinguish between the ideal process ($\mathsf{Z}(1)$) and the real world ($\mathsf{Z}(p+1)$) with non-negligible probability ϵ; this implies $|\Pr[\mathsf{Z}(1)=0]-\Pr[\mathsf{Z}(p+1)=0]|>\epsilon$. Without loss of generality, let us say that in fact

$$\Pr[\mathsf{Z}(1)=0]-\Pr[\mathsf{Z}(p+1)=0]>\epsilon.$$

This implies there exists a ϕ' such that

$$\Pr[\mathsf{Z}(\phi')=0]-\Pr[\mathsf{Z}(\phi'+1)=0]>\frac{\epsilon}{np}.$$

Assume G guesses (j,h) such that $m_{(j,h)}$ is the ϕ'th message. This means that if G was given back $c_{\phi'}=enc(e,m_0)$, then G will output $\mathsf{Z}(\phi')$. Conversely, G will output $\mathsf{Z}(\phi'+1)$ if $c_{\phi'}=enc(e,m_{(j,h)})$. Thus when $m_{(j,h)}$ is the ϕ'th message, G will output $b'=b$ with probability $\frac{1}{2}+\frac{\epsilon}{2np}$. Because j,h are random, we hit ϕ' with probability $\frac{1}{np}$ and so G guesses b with probability $\frac{1}{2}+\frac{\epsilon}{2(np)^2}$, which is a non-negligible advantage.

The existence of G is a contradiction of S's IND-CCA2 security. It must then be that there does not exist an Z that is able to distinguish between interactions in the ideal process with F_{PKE} and the real world with π_S. Thus, for a local, stateless IND-CCA2 secure encryption scheme S, π_S securely realizes F_{PKE}. $\square$

6.2 Signatures

As before, the adversary's powers are strictly weaker in our new formulation of F_{SIG} (Figure 5-3) as compared to the previous one (Figure 4-3 from [15]), implying that a local stateless scheme must be at least EUF-ACMA to realize F_{SIG}. In this case, however, we can go one step further and show that any realizing scheme must be at least a strong signature scheme. We also show that strong signature schemes may be used to securely realize this new formulation of F_{SIG}.

Definition 20 *Define protocol π_Σ using signature scheme Σ as follows*

1. *When activated, within some P_i and with input* (KeyGen, id)*, run algorithm gen, output the verification key v and record the signing key s.*

2. *When activated, within P_i and with input* (Sign, id, m)*, return $sig(s, m)$.*

3. *When activated, within some party P_j and with input* $(\mathsf{Verify}, id, v', m, \sigma)$*, return $ver(v', m, \sigma)$.*

Theorem 5 *Let $\Sigma = (gen, sig, ver)$ be a signature scheme over domain $\mathcal{D}$ composed of local, stateless algorithms. Then π_Σ securely realizes F_{SIG} with respect to domain $\mathcal{D}$ and non-adaptive adversaries if and only if Σ is strong (Definition 6).*

Note that neither the F_{SIG} formulation nor the strong security definition requires that verification be a deterministic process. We allow for probabilistic verification algorithms in order to give as general a theorem statement as possible – in practice, however, deterministic verification algorithms are almost always used. Strong security itself, does imply a sort of verification "consistency" already; if it were possible for an adversary to produce a message and signature where *ver* returned both 0 and 1 with non-negligible probabilities, then either Σ is not complete or an adversary is able to violate Σ's strict unforgeability.

Proof. "Only if" - Given a Σ that is not strong, we will construct an environment Z that can distinguish whether it is interacting with A and π_Σ or S and F_{SIG}.

1. Assume Σ is not complete, i.e. there exists $m \in M_k$ such that $\Pr[(s, v) \leftarrow gen(1^k);\ \sigma \leftarrow sig(s, m) : 0 \leftarrow ver(m, \sigma, v)] > neg(k)$ for infinitely many k's. In this case, Z sets $sid = (\mathsf{P}, 0)$ and activates some uncorrupted party P with input (KeyGen, sid), followed by (Sign, sid, m). After obtaining v and σ, it then activates some other party P' with $(\mathsf{Verify}, sid, m, \sigma, v)$ and outputs the returned verification value.

   When interacting with the ideal process, Z clearly always outputs 1, whereas in the interaction with π_Σ, Z outputs 0 with non-negligible property. Thus, a π_Σ that realizes F_{SIG} must use a Σ that exhibits completeness.

2. Assume Σ is not strictly unforgeable; i.e. there exists a PPT forger G that can produces a valid (m, σ) previously unknown to it. Z runs an internal copy of G and hands it the public key v obtained from an uncorrupted party P. Whenever G asks

its oracle to sign a message m, Z activates P with input $(\mathsf{Sign}, sid = (\mathsf{P}, 0), m)$, and reports the output to G. When G generates a pair (m, σ), Z proceeds as follows:

(a) If (m, σ) was a previous response to G's oracle queries, output 0 and halt.

(b) Otherwise, activate some other uncorrupted party with input $(\mathsf{Verify}, sid, m, \sigma)$ and output the verification result.

Because v belongs to an uncorrupted party, when Z interacts with the ideal process, it will always output 0. By the definition of G, when Z interacts with π_Σ in the concrete world, it will output 1 with some non-negligible probability. Thus, a π_Σ that realizes F_{SIG} must use a Σ that exhibits strict unforgeability.

Therefore, a π_Σ realizes F_{SIG} only if Σ is strong.

"If" - Assume Σ is strong but π_Σ does not realize F_{SIG}; i.e. there is a real-life adversary A such that for any ideal-process adversary S there exists an environment Z that can distinguish between the interactions of A and π_Σ or S and F_{SIG}. Since Z succeeds for any S, it must succeed for the following S:

- S runs a simulated copy of A, denoted A_S.
- Any input from Z is forwarded to A_S. A_S's outputs are copied to S's outputs.
- When S receives a message $(\mathsf{KeyGen}, sid, \mathsf{P})$ from F_{SIG}, it checks if sid is of the form (P, sid'). If it is not, S ignores the request. If it is, S runs $(s, v) \leftarrow gen(1^k)$, records s, and returns $(\mathsf{VerificationKey}, sid, v)$ along with algorithms $\mathsf{S} = sig(s, \cdot)$ and $\mathsf{V} = ver(\cdot, \cdot, \cdot)$ to F_{SIG}.
- When A_S tries to corrupt some party P, S corrupts P in the ideal process. If P is the signer, then S reveals the signing key s (and potentially the internal state of algorithm sig) as the internal state of P.

Assume Σ is complete (otherwise the theorem is proven). We consider how the environment might distinguish between the ideal and concrete world. There are two cases:

1. a party P_0 outputs a new valid signature under another honest party P_1's verification key without ever having been told the new signature value or

2. a signature for a message that was never signed by an honest party P verifies under the key associated with P.

(1) This could never happen in the ideal process, since F_{SIG} will never verify a forged signature under an uncorrupted verification key, so it must be this event happens with non-negligible probability during Z's interaction with the concrete world. P_0 outputs the new signature on the behest of either Z or A_G – if done on the behalf of Z, then this is no different than situation (2), and we thus only consider if A_G created this signature. If it did, then A, with adaptive access to a signing oracle, is capable of producing a new valid signature under a desired verification key with non-negligible probability. This violates Σ's strict unforgeability, so this must not be the case.

(2) This too could never happen in the ideal process, so it must be that the event happens with non-negligible probability in the concrete world. We construct the following PPT algorithm G:

- G runs a simulated copy of Z and simulates for Z an interaction with S in the ideal process with F_{SIG} (and some encryption functionality). Note that G plays the role of both S and F_{SIG} for simulated Z.

- Instead of actually simulating S, G runs a simulated copy of A and forwards its inputs/outputs accordingly. We label it A_G.

- When S is asked to generate a verification key, instead of running gen, G gives A_G and the simulated F_{SIG} the key, v, under which a forgery is desired.

- When the simulated F_{SIG} is asked to sign m by the proper party, G asks its oracle to sign m and then has F_{SIG} return the obtained σ.

- Whenever the simulated Z activates some uncorrupted party with input $(\mathsf{Verify}, sid, m, \sigma, v)$, G checks whether (m, σ) is a new valid signature pair previously unseen. If it is, G outputs the pair and halts; if not, it continues the simulation. If A_G asks to corrupt the signer then G halts with a failure output.

Consider that A_G outputs a new valid (m, σ) pair. If it doesn't then by Σ's completeness, the nature of the ideal encryption's realization, and our analysis of situation (1), Z's view of an interaction with A and π_Σ is statistically close to its view of an interaction with

S and F_{SIG}. Since we know Z distinguishes between these two cases with non-negligible probability, it must be that, with non-negligible probability, A_{G} asks for the signature on a previously unseen signature pair that verifies. In this case G outputs a violation of $\Sigma' s$ strict unforgeability with non-negligible probability. Since Σ is strong, this must not be the case.

It then follows that the environment is unable to meaningfully distinguish between the ideal and real worlds and thus π_Σ realizes F_{SIG} if Σ is strong.

$\square$

Note that in the "if" direction, we allowed for adaptive adversaries, but the "only if" direction required non-adaptive adversaries. This implies that if Σ is strong then π_Σ securely realizes F_{SIG} against adaptive adversaries (but not vice versa).

Chapter 7

Adding Signatures to Universally Composable Symbolic Analysis

Having redefined the ideal functionalities as needed, we reconstruct the UCSA framework now expanded to include digital signatures.

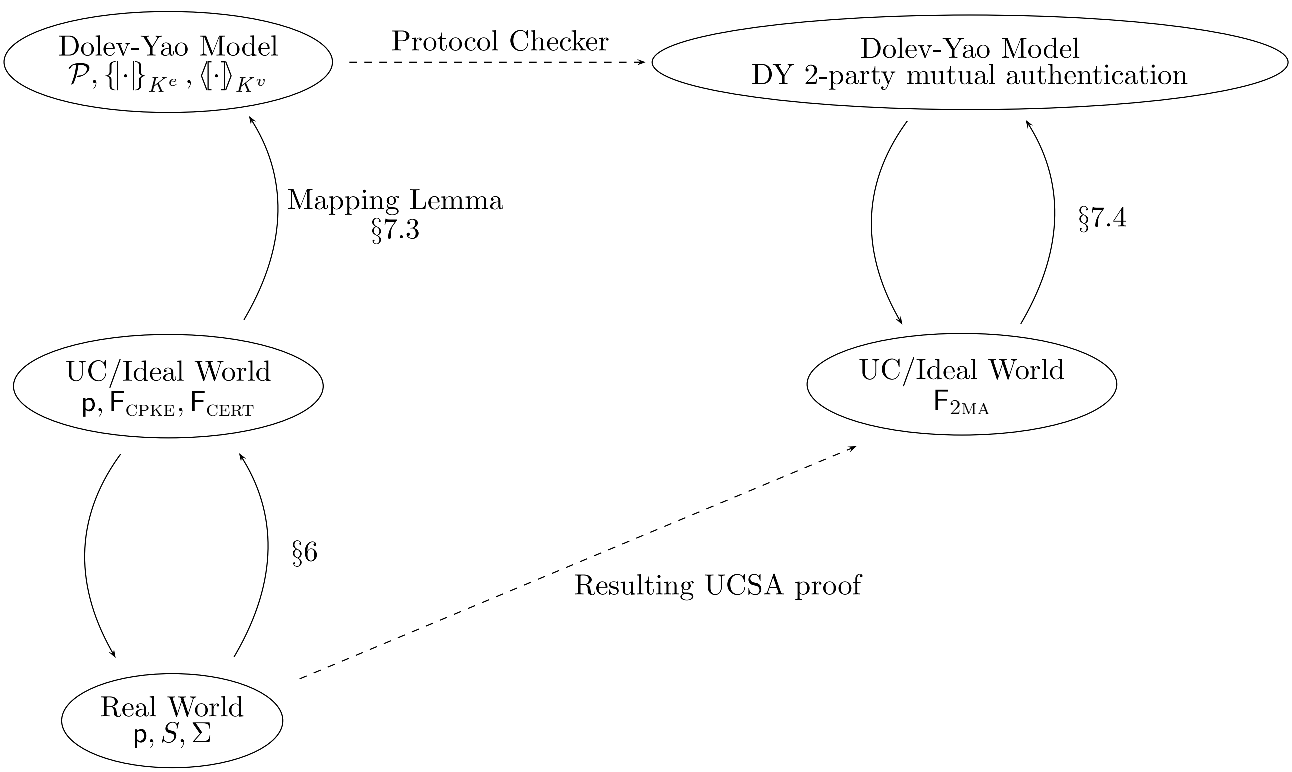

Figure 7-1: A graphical representation of the UCSA framework with signatures

7.1 Dolev-Yao Algebra

The Dolev-Yao algebra is redefined to include the symbols necessary for signatures.

Definition 21 (The Dolev-Yao Message Algebra) *Messages in the Dolev-Yao algebra $\mathcal{A}$ are composed of atomic elements of the following types:*

- *Party identifiers ($\mathcal{M}$) – These are denoted by symbols $P_1, P_2, ..$ for a finite number of names in the algebra. These are public and are associated with a* role *which is either that of Initiator or Responder.*
- *Nonces ($\mathcal{R}$) – These can be thought of as a finite number of private, unpredictable random-strings. These symbols are denoted by $R_1, R_2, ...$ and so on.*
- *Public keys ($\mathcal{K}_{Pub}$) – These are denoted by symbols $K^e_{P_1}, K^e_{P_2}, ...$ which are public and each associated with a particular party identifier.*
- *Verification keys ($\mathcal{K}_{Ver}$) – These are denoted by symbols $K^v_{P_1}, K^v_{P_2}, ...$ which are public and each associated with a particular party identifier.*
- *A garbage term, written $\mathcal{G}$, to represent ill-formed messages,*
- *$\perp$, to represent an error or failure,*
- *Starting, to indicate that a protocol execution has begun, and*
- *Finished, to indicate that a protocol execution has ended.*

Messages in the algebra can be compounded by the following symbolic operations:

- *pair: $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$. When messages m and m' are paired, we write $m|m'$.*
- *encrypt : $\mathcal{K}_{Pub} \times \mathcal{A} \rightarrow \mathcal{A}$. When message m is encrypted with public key K^e_P, we write $\{\!|m|\!\}_{K^e_P}$*
- *sign : $\mathcal{K}_{Ver} \times \mathcal{A} \rightarrow \mathcal{A}$ When message m is signed for verification key K^v_P, we write $\langle\!|m|\!\rangle_{K^v_P}$*

.

Definition 22 (Closure) *Let*

- $\mathcal{R}_{Adv} \subset \mathcal{R}$ *be the set of nonces associated with the adversary,*
- $\mathcal{K}^e_{Adv} = \{K^e_P : P \in \mathcal{M}_{Adv}\}$ *be the set of encryption keys belonging to corrupted parties* ($\mathcal{K}^e_{Adv} \subset \mathcal{K}_{Pub}$)*, and*
- $\mathcal{K}^v_{Adv} = \{K^v_P : P \in \mathcal{M}_{Adv}\}$ *be the set of verification keys belonging to corrupted parties* ($\mathcal{K}^v_{Adv} \subset \mathcal{K}_{Ver}$)*.*

Then the closure of a set $S \in \mathcal{A}$*, written* $C[S]$*, is the smallest subset of* $\mathcal{A}$ *such that:*

1. $S \subseteq C[S]$,
2. $\mathcal{M} \cup \mathcal{K}_{Ver} \cup \mathcal{K}_{Pub} \cup \mathcal{R}_{Adv} \subseteq C[S]$,
3. *If* $\{\!|m|\!\}_K \in C[S]$ *and* $K \in \mathcal{K}^e_{Adv}$*, then* $m \in C[S]$,
4. *If* $m \in C[S]$ *and* $K \in \mathcal{K}_{Pub}$*, then* $\{\!|m|\!\}_K \in C[S]$,
5. *If* $\langle\!|m|\!\rangle_K \in C[S]$ *then* $m \in C[S]$, [1]
6. *If* $m \in C[S]$ *and* $K \in \mathcal{K}^v_{Adv}$*, then* $\langle\!|m|\!\rangle_K \in C[S]$,
7. *If* $m|m' \in C[S]$*, then* $m \in C[S]$ *and* $m' \in C[S]$*, and*
8. *If* $m \in C[S]$ *and* $m' \in C[S]$*, then* $m|m' \in C[S]$.

The algebra remains free under these changes. Figure 7-2 shows an updated example symbolic message parse tree.

Definition 23 (Dolev-Yao Trace) *We inductively define a Dolev-Yao trace t for protocol* $\mathcal{P}$ *as a description of events that occur during the execution of* $\mathcal{P}$*.*

$$t = H_0\ H_1\ H_2\ \ldots\ H_{n-2}\ H_{n_1}\ H_n$$

where event H_i *is either*

- *of the form* [*"input"*$, P, o_i, P', S$]*, that indicates the initial input of participant* P *to take the role* o_i *and interact with participant* P'*, assuming initial internal state* S*.*

[1] As explained in Section 2.2.2, we only consider signature schemes that are message revealing or, alternatively, only valid when transmitted with their message.

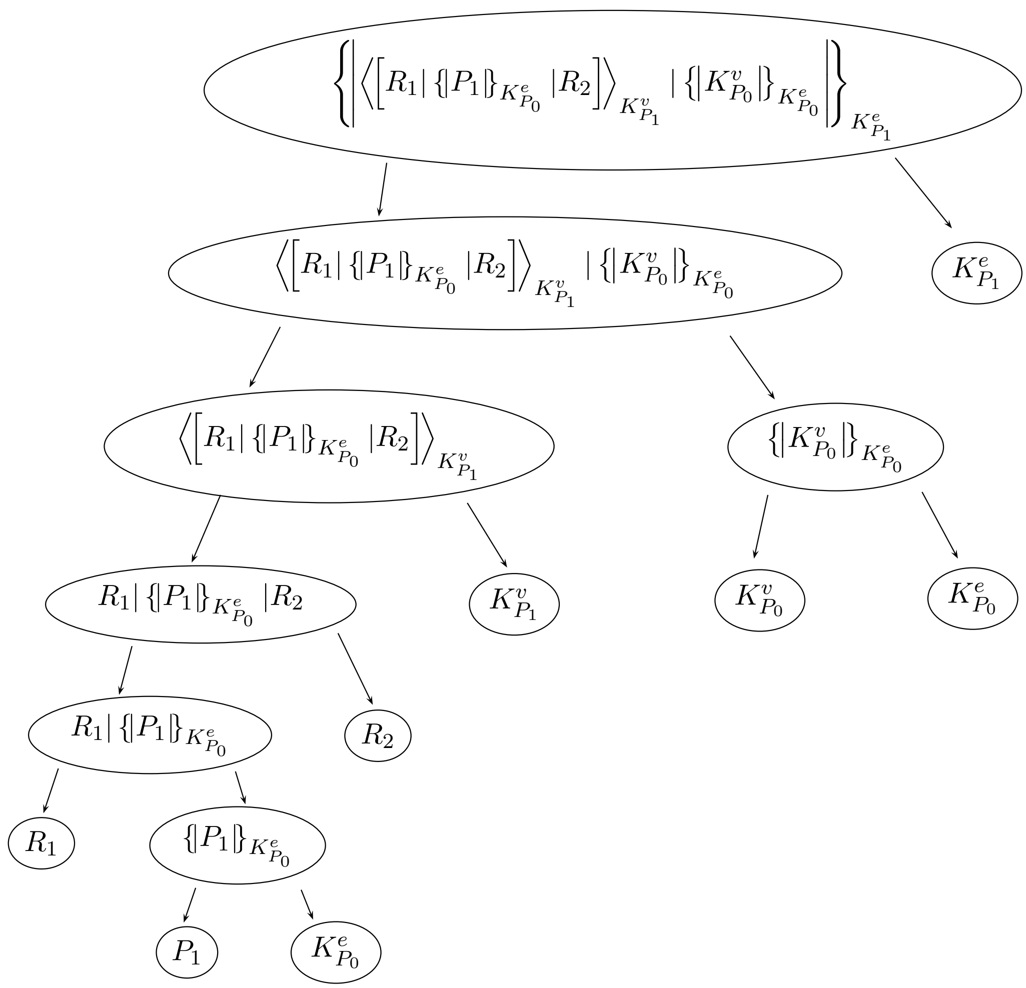

Figure 7-2: Example Dolev-Yao parse tree with signatures

- *an adversary event (where $j, k < i$) of the form*
  - [*"enc"*$, j, k, m_i$], *in which case* $m_k \in \mathcal{K}_{Pub}$ *and* $m_i = \{\!|m_j|\!\}_{m_k}$,
  - [*"dec"*$, j, k, m_i$], *in which case* $m_k \in \mathcal{K}^e_{Adv}$, *and* $m_j = \{\!|m_i|\!\}_{m_k}$,
  - [*"sign"*$, j, k, m_i$], *in which case* $m_k \in \mathcal{K}^v_{Adv}$ *and* $m_i = \langle\!|m_j|\!\rangle_{m_k}$,
  - [*"pair"*$, j, k, m_i$], *in which case* $m_i = m_j|m_k$
  - [*"extract-l"*$, j, m_i$], *in which case* $m_j = m_i|m_k$ *for some* $m_k \in \mathcal{A}$,
  - [*"extract-r"*$, j, m_i$], *in which case* $m_j = m_k|m_i$ *for some* $m_k \in \mathcal{A}$,
  - [*"random"*$, m_i$], *in which case* $m_i = R$ *for some* $R \in \mathcal{R}_{Adv}$,
  - [*"name"*$, m_i$], *in which case* $m_i = A$ *for some* $A \in \mathcal{M}$,
  - [*"pubkey"*$, m_i$], *in which case* $m_i = K$ *for some* $K \in \mathcal{K}_{Pub}$,
  - [*"deliver"*$, j, P_i$], *in which case the message* m_j *is delivered to party* P_i.
- *or a participant event of the form* [*"output"*$, P_i, m_i$] *or* [*"message"*$, P_i, m_i$], *in that case* [*"deliver"*$, k, p_i$] *is the most recent adversary event in the trace (for some* k*) and the protocol action for* P_i *in its current role and internal state, upon receiving* m_k, *is to output/send message* m_i. *(* $\mathcal{P}(S_j, o_i, m_k, P_i) \rightarrow (S_i, m_i, \{output, message\})$ *).*

Note that there is no adversarial action for signature verification, this is because in the symbolic world, an adversary is either able to create the signature symbol or not – there is no need to verify that a signature symbol is a signature symbol.

7.2 Simple Protocols

Simple protocols and their mappings are redefined to allow signing capabilities.

Definition 24 (Simple protocols) *A* simple protocol *is a pair of interactive Turing machines (ITMs)* $\{\mathtt{M}_1, \mathtt{M}_2\}$, *one for each role, where each machine* $\mathtt{M}_i$ *implements an algorithm described by a pair* (Σ, Π)*:*

- Σ *is a* store, *a mapping from variables to tagged values (explained further below) and*
- Π *is a program that expects as input*
  - *The security parameter* k,

Π ::= BEGIN; STATEMENTLIST

BEGIN ::= `input`$(\mathsf{SID}, \mathsf{RID}, \mathsf{PID}_0, \mathsf{PID}_1, \mathsf{RID}_2, ...)$;
(Store $\langle$"role", $\mathsf{RID}\rangle$, $\langle$"name", $\mathsf{PID}_0\rangle$, $\langle$"name", $\mathsf{PID}_1\rangle$, $\langle$"name", $\mathsf{PID}_2\rangle$,...
in local variables MyRole, MyName, $\mathsf{PeerName}$, $\mathsf{OtherName}_2$,...
respectively.

STATEMENTLIST ::= STATEMENT STATEMENTLIST
| FINISH

STATEMENT ::= `newrandom`(v)
(generate a k-bit random string r and store $\langle$"random", $r\rangle$ in v)
| `encrypt`$(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
(Send $(\texttt{Encrypt}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \mathsf{v2})$ to F_{CPKE} where $v1 = \langle$"pid", $\mathsf{PID}\rangle$,
receive c, and store $\langle$"ciphertext", c, $\langle \mathsf{PID}_1, \mathsf{SID}\rangle\rangle$ in $\mathsf{v3}$)
| `decrypt`$(\mathsf{v1}, \mathsf{v2})$
(If the value of $\mathsf{v1}$ is $\langle$"ciphertext", $c'\rangle$ then send
$(\texttt{Decrypt}, \langle \mathsf{PID}_0, \mathsf{SID}\rangle, c')$ to F_{CPKE} instance $\langle \mathsf{PID}_0, \mathsf{SID}\rangle$,
receive some value m, and store m in $\mathsf{v2}$ Otherwise, end.
| `sign`$(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
(Send $(\texttt{Sign}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \mathsf{v2})$ to F_{CERT} where $v1 = \langle$"pid", $\mathsf{PID}\rangle$,
receive σ, and store $\langle$"signature", σ, $\langle \mathsf{PID}_1, \mathsf{SID}\rangle\rangle$ in $\mathsf{v3}$)
| `verify`$(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
(If the value of $\mathsf{v1}$ is $\langle$"signature", $\sigma'\rangle$ then send
$(\texttt{Verify}, \langle \mathsf{PID}_0, \mathsf{SID}\rangle, \sigma', \mathsf{v2})$ to F_{CERT} instance $\langle \mathsf{PID}_0, \mathsf{SID}\rangle$,
receive some value b, and store b in $\mathsf{v3}$
Otherwise, end.
| `send`(v)
(Send value of variable v)
| `receive`(v)
(Receive message, store in v)
| `output`(v)
(send value of v to local output)
| `pair`$(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
(Store $\langle$"pair", $\sigma_1, \sigma_2\rangle$ in $v3$, where σ_1 and σ_2 are the values of
$\mathsf{v1}$ and $\mathsf{v2}$, respectively.)
| `separate`$(\mathsf{v1}, \mathsf{v2}, \mathsf{v3})$
(if the value of $v1$ is $\langle$"join", $\sigma_1, \sigma_2\rangle$, store σ_1 in $v2$
and σ_2 in $v3$ (else end))
| `if` $(\mathsf{v1} == \mathsf{v2}$ `then` STATEMENTLIST `else` STATEMENTLIST
(where $\mathsf{v1}$ and $\mathsf{v2}$ are compared by value, not reference)

FINISH ::= `output`$(\langle$"finished", $\mathsf{v}\rangle)$; `end`.

The symbols v, $\mathsf{v1}$, $\mathsf{v2}$ and $\mathsf{v3}$ represent program variables. It is assumed that $\langle$"pair", $\sigma_1, \sigma_2\rangle$ encodes the bit-strings σ_1 and σ_2 in such a way that they can be uniquely and efficiently recovered. A party's input includes its own PID, the PID of its peer, and other PIDs in the system. Recall that the SID of an instance of F_{CPKE} is an encoding $\langle \mathsf{PID}, \mathsf{SID}\rangle$ of the PID and SID of the legitimate recipient.

Figure 7-3: The grammar of simple protocols

– *Its SID* SID*, its PID* PID*, and its RID* RID*,*

– PID_1 *that represents the name for the other participant of this protocol execution.*

The programs tags the input values, binds them to variables in the store, and then acts according to a sequence of commands consistent with the grammar in Figure 3-2.

Definition 25 (Mapping of simple protocols to symbolic protocols) *Let* $\mathsf{p} = \{\mathtt{M}_0, \mathtt{M}_1\}$ *be a simple protocol. Then* $\widehat{\mathsf{p}}$ *is the Dolev-Yao protocol*

$$\mathcal{P}_i : \mathcal{S} \times \mathcal{M} \times \mathcal{O} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathcal{A} \times message \times \mathcal{A} \times output$$

that implements ITM $\mathtt{M}$*, except that:*

- *The variables of* $\mathtt{M}$ *are interpreted as elements of the symbolic message algebra* $\mathcal{A}$*.*

- *Instead of receiving as input* SID*,* PID_0*,* PID_1*,* RID*, the store is initialized with its own name* P_0*, its own keys* $K^e_{P_0}|K^v_{P_0}$*, and a name* P_1 *and keys* $K_{P^e_1}|K^v_{P_1}$ *of the other participant. The symbols* P_0 *and* P_1 *represent* PID_0 *and* PID_1*, respectively. Similarly, the symbols* K_0 *and* K_1 *represent* $\langle \mathsf{PID}_0, \mathsf{SID} \rangle$ *and* $\langle \mathsf{PID}_1, \mathsf{SID} \rangle$*, respectively.*

- *Instead of creating a new random bit-string, the symbolic protocol returns* $R_{(i,n)}$ *and increments* n *(which starts at 0),*

- *Instead of sending* $(\mathtt{Encrypt}, \langle \mathsf{PID}, \mathsf{SID} \rangle, \mathsf{M})$ *to* F_{CPKE} *and storing the result, the composed symbol* $\{\!|\Sigma(\mathsf{M})|\!\}_{K_{P_1}}$ *is stored instead (where* $\Sigma(\mathsf{M})$ *is the value bound to the variable* M *in the store* Σ*).*

- *Instead of sending* $(\mathtt{Decrypt}, \langle \mathsf{PID}_0, \mathsf{SID} \rangle, \mathsf{C})$ *to* F_{CPKE} *and storing the result, the value stored depends on the form of* $\Sigma(\mathsf{C})$*. If* $\Sigma(\mathsf{C})$ *is of the form* $\{\!|M|\!\}_{K_{P_0}}$ *then the value* M *is stored. Otherwise, the garbage value* $\mathcal{G}$ *is stored instead.*

- *Instead of sending* $(\mathtt{Sign}, \langle \mathsf{PID}, \mathsf{SID} \rangle, \mathsf{M})$ *to* F_{CERT} *and storing the result, the composed symbol* $\langle\!|\Sigma(\mathsf{M})|\!\rangle_{K_{P_0}}$ *is stored instead (where* $\Sigma(\mathsf{M})$ *is the value bound to the variable* M *in the store* Σ*).*

- *Instead of sending* $(\mathtt{Verify}, \langle \mathsf{PID}_0, \mathsf{SID} \rangle, \mathsf{S}, \mathsf{M})$ *to* F_{CERT} *and storing the result, the value stored depends on the form of* $\Sigma(\mathsf{S})$*. If* $\Sigma(\mathsf{S})$ *is of the form* $\langle\!|M|\!\rangle_{K_{P_1}}$ *then the value* 1 *is stored. Otherwise,* 0 *is stored instead.*

- *Pairing and separation use the symbolic pairing operator.*

- *Lastly, the bit-strings "starting" and "finished" are mapped to the Dolev-Yao symbols Starting and Finished, respectively.*

7.3 UC to Dolev-Yao: The Revised Mapping Lemma

Definition 26 (Traces of concrete protocols) *Let* p *be a* F*-hybrid protocol. Inductively define* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$*, as the trace of protocol* p *in conjunction with adversary* A *and environment* Z *with inputs* $z, \vec{r}$*, and security parameter* k*. Initially, the trace is the null string. The trace then grows as the protocol's execution progresses.*

- *If the environment provides input* m *to a party with id* $(\mathsf{SID}, \mathsf{RID})$*, then* $\langle$ *"input"*$, (\mathsf{SID}, \mathsf{RID}), m\rangle$ *is appended to the end of* t*.*

- *If the adversary provides input* m *to a party with id* PID*, then* $\langle$ *"adv"*$, \mathsf{PID}, m\rangle$ *is appended to the end of* t*.*

- *If a party* PID *generates a new random string* r*, then* $\langle$ *"random"*$, r\rangle$ *is appended to* t*.*

- *If a party pairs values* m_1 *and* m_2 *to form* (m_1, m_2)*, then* $\langle$ *"pair"*$, m_1, m_2\rangle$ *is appended to* t*.*

- *If a party* PID *writes a message* m*, it does so in one of two ways*

  - *if it writes* m *on its local output tape, then* $\langle$ *"output"*$, \mathsf{PID}, m\rangle$ *is appended to* t*.*

  - *if it writes* m *on its outgoing communication tape, then* $\langle$ *"message"*$, \mathsf{PID}, m\rangle$ *is appended to* t*.*

- *If* F_{CPKE} *is activated by party* PID *with call* $(\mathsf{Encrypt}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m)$ *and* F_{PKE} *responds with ciphertext* c*, then* $\langle$ *"ciphertext"*$, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *is appended to* t*. (If* F_{CPKE} *returns,* $\perp$ *then nothing is appended to* t*).*

- *If* F_{CPKE} *is activated by party* PID *with call* $(\mathsf{Decrypt}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c)$ *and* F_{CPKE} *responds with plaintext* m*, then* $\langle$ *"dec"*$, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$ *is appended to* t*. (If* F_{CPKE} *returns,* $\perp$ *then nothing is appended to* t*).*

- *If* F_{CERT} *is activated by party* PID *with call* $(\mathsf{Sign}, \langle \mathsf{PID}, \mathsf{SID} \rangle, m)$ *and* F_{CPKE} *responds with signature* σ*, then* $\langle$ *"signature"*$, \langle \mathsf{PID}, \mathsf{SID} \rangle, m, \sigma \rangle$ *is appended to* t*. (If* F_{CERT} *returns,* $\perp$ *then nothing is appended to* t*).*

- *If* F_{CERT} *is activated by party* PID *with call* $(\mathsf{Verify}, \langle \mathsf{PID}, \mathsf{SID} \rangle, m, \sigma)$ *and* F_{CERT} *responds with boolean bit b, then* $\langle$ *"ver"*$, \langle \mathsf{PID}, \mathsf{SID} \rangle, m, \sigma, b \rangle$ *is appended to* t*. (If* F_{CERT} *returns,* $\perp$ *then nothing is appended to* t*).*

$\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ *denotes* t *upon completion of the protocol execution. Let* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)$ *denote the random variable for* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z, \vec{r})$ *when* $\vec{r}$ *is uniformly chosen. Let* $\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}$ *denote the probability ensemble* $\{\text{TRACE}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k, z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*}$

Definition 27 (The mapping from concrete traces to symbolic traces) *Let* p *be a concrete* $\mathsf{F}_{\text{CPKE}}/\mathsf{F}_{\text{CERT}}$*-hybrid protocol and let* t *be a trace of an execution of* p *with security parameter* k*, environment* Z *with input* z*, and random input vector* $\vec{r}$*. We determine the mapping of* t *to a Dolev-Yao trace in two steps. (These steps can be thought of as two "passes" on the string* t*.)*

(I.) *First, we read through the string* t *character by character, in order, and inductively define the following partial mapping* f *from* $\{0,1\}^*$ *to elements of the algebra* $\mathcal{A}$*. (Note that the patterns in* t *addressed below may be nested and overlapping. That is, the same substring may be part of multiple patterns. A pattern is recognized as soon as the last character in the pattern is read.)*

- *Whenever we encounter a pattern of the form* $\langle$ *"name"*$, \beta \rangle$ *for some string* β *and* $f(\langle$ *"name"*$, \beta \rangle)$*is not yet defined then set* $f(\langle$ *"name"*$, \beta \rangle) = P$ *for some new symbol* $P \in \mathcal{M}$ *not in the range of* f *so far.*

- *Whenever we encounter a pattern of the form* $\langle$ *"boolean"*$, \beta \rangle$ *for some string* β *and* $f(\langle$ *"boolean"*$, \beta \rangle)$ *is not yet defined then set* $f(\langle$ *"boolean"*$, \beta \rangle) = P$ *for some new symbol* $P \in \mathcal{B}$ *not in the range of* f *so far.*

- *Whenever we encounter in some event a pattern of the form* $\langle$ *"random"*$, \beta \rangle$ *for some string* β *and* $f(\langle$ *"random"*$, \beta \rangle)$ *is not yet defined then set* $f(\langle$ *"random"*$, \beta \rangle) = N$ *for some new symbol* $N \in \mathcal{R}$ *that is not in the range of* f *so far.*

- *Whenever we encounter a pattern of the form* $\langle \langle \text{"pid"}, \mathsf{PID} \rangle, \langle \text{"sid"}, \mathsf{SID} \rangle \rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}$*, with* $f(\langle$ *"pubkey"*$, \langle \mathsf{PID}, \mathsf{SID} \rangle \rangle)$ *and/or* $f(\langle$ *"verkey"*$, \langle \mathsf{PID}, \mathsf{SID} \rangle \rangle)$ *not* *yet defined, then set* $f(\langle\text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle) = K$ *and/or* $f(\langle\text{"verkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle) = K'$ *for some new* $K \in \mathcal{K}_{Pub}$ *and* $K' \in \mathcal{K}_{Ver}$ *not already in the range of* f*.*

- *Whenever we encounter a pattern of the form* $\langle\text{"pair"}, \beta_1, \beta_2\rangle$*, then proceed as follows. First, if* $f(\beta_1)$ *is not yet defined then set* $f(\beta_1) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Similarly, if* $f(\beta_2)$ *is not yet defined then set* $f(\beta_2) = \mathcal{G}$*. Finally, set* $f(\langle\text{"pair"}, \beta_1, \beta_2\rangle) = f(\beta_1)|f(\beta 2)$*.*

- *Whenever we encounter a pattern of the form* $\langle\text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}, m, c$*, then* f *is expanded so that* $f(\langle\text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle\text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever an encryption call to* F_{CPKE} *is made. Also, at this point both* $f(m)$ *and* $f(\langle\text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *must already be defined, since this is an encryption call made by a party running a simple protocol.)*

- *Whenever we encounter a pattern of the form* $\langle\text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then proceed as follows. First, if* $f(m)$ *is not yet defined, then set* $f(m) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Next, set* $f(\langle\text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c\rangle) = \{\!|f(m)|\!\}_{f(\langle\text{"pubkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever a decryption call to* F_{CPKE} *is made. The case where* $f(m) = \mathcal{G}$ *occurs when a ciphertext was not generated via the encryption algorithm. It includes both the case where the decryption algorithm fails and the case where the decryption algorithm outputs a message that cannot be parsed by simple protocols.)*

- *Whenever we encounter a pattern of the form* $\langle\text{"signature"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, \sigma\rangle$ *for some strings* $\mathsf{PID}, \mathsf{SID}, m, \sigma$*, then* f *is expanded so that* $f(\langle\text{"signature"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \sigma\rangle) = \langle\!|f(m)|\!\rangle_{f(\langle\text{"verkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*. (Recall that such a pattern is generated whenever a signature call to* F_{CERT} *is made. Also, at this point both* $f(m)$ *and* $f(\langle\text{"verkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)$ *must already be defined, since this is a signing call made by a party running a simple protocol.*

- *Whenever we encounter a pattern of the form* $\langle\text{"verification"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \sigma, m, b\rangle$*, then proceed as follows. First, if* $f(m)$ *is not yet defined, then set* $f(m) = \mathcal{G}$*, where* $\mathcal{G}$ *is the garbage symbol. Next, if* $b = 1$*, set* $f(\langle\text{"verify"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \sigma\rangle) = \langle\!|f(m)|\!\rangle_{f(\langle\text{"verkey"}, \langle \mathsf{PID}, \mathsf{SID}\rangle\rangle)}$*, else set* $f(\langle\text{"verify"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \sigma\rangle) = \mathcal{G}$*. (Recall that such a pattern is generated* *whenever a signature call to* F_{CERT} *is made. The case where* $f(m) = \mathcal{G}$ *occurs when a signature was not generated via the signing algorithm. If the signature is valid,* σ *is made a valid signature symbol, if it isn't valid, it is set to garbage)*

(II.) *In the second step, we construct the actual Dolev-Yao trace. Let* $\mathsf{t} = \mathsf{G}_1||\mathsf{G}_2||\ldots\mathsf{t}_n$ *be the concrete trace. Then construct the Dolev-Yao trace* $\widehat{\mathsf{t}}$ *by processing each* G *in turn, as follows:*

- *If* $\mathsf{G}_i = \langle \text{"input"}, (\mathsf{SID}, \mathsf{RID}), m\rangle$*, then we find* $\mathsf{m} = f(m)$*, and generate the symbolic event* $H = [\text{"input"}, P, \mathsf{m}]$ *(where* P *is the symbolic name of the input recipient).*

- *If* $\mathsf{G}_i = \langle \text{"ciphertext"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, c\rangle$ *or* $\mathsf{G}_i = \langle \text{"dec"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, c, m\rangle$*, then no symbolic event is generated.*

- *If* $\mathsf{G}_i = \langle \text{"signature"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, m, \sigma\rangle$ *or* $\mathsf{G}_i = \langle \text{"ver"}, \langle \mathsf{PID}, \mathsf{SID}\rangle, \sigma, m, b\rangle$*, then no symbolic event is generated.*

- *If* $\mathsf{G}_i = \langle \text{"output"}, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle \text{"name"}, \mathsf{PID}\rangle), output, f(m)).$$

- *If* $\mathsf{G}_i = \langle \text{"message"}, \mathsf{PID}, m\rangle$ *then* G_i *is mapped to the symbolic participant event*

$$(f(\langle \text{"name"}, \mathsf{PID}\rangle), message, f(m)).$$

- *If* $\mathsf{G} = \langle \text{"adv"}, \mathsf{PID}, m\rangle$*, let* $\mathsf{m} = f(m)$*. Then there are two cases:*

  1. m *is in the closure of the symbolic interpretations of the messages sent by the parties in the execution so far, i.e.*

  $$\mathsf{m} \in C\left[\{\mathsf{m}' : \mathsf{m}' = f(m') \text{ and the event } \langle \text{"message"}, \mathsf{PID}, m'\rangle \text{ is a prior event in } t\}\right].$$

  In this case there exists a finite sequence of adversary events that produces m_i*. Then* G *is mapped to this sequence of events* H_{i_1}*,* $H_{i,2}\ldots H_{i,n'}$ *so that the message of* $H_{i,n'-1}$ *is* m_i *and* $H_{i,n'} = [\text{"deliver"}, (i, n'-1), P']$ *(where* P' *is the Dolev-Yao name of the concrete participant who received the message from the concrete adversary).*

2. *Otherwise,* m *is not in the above closure. In this case,* G *maps to the Dolev-Yao event* [*"fail"*, m_i].

We now prove the Mapping Lemma for public key encryption and digital signatures using the reformulated ideal functionalities of Chapter 5.

Lemma 6 (The Revised Mapping Lemma) *Let* F *denote both ideal functionalities* F_{CPKE} *and* F_{CERT}*. For all simple* F*-hybrid protocols* p*, adversaries* A*, environments* Z*, and inputs* z *of length polynomial in the security parameter* k*,*

$$\Pr\left[\mathsf{t} \leftarrow \text{TRACE}^{\mathsf{F}}_{\mathsf{p},\mathsf{A},\mathsf{Z}}(k,z) : \widehat{\mathsf{t}} \textit{ is a valid DY trace for } \widehat{\mathsf{p}}\right] \geq 1 - \mathit{neg}(k)$$

Proof. Let t be the trace of a simple protocol p. We will show two things: (1) if $\widehat{\mathsf{t}}$ does not contain the event ["fail", m_i], then it is a valid DY trace of protocol $\widehat{\mathsf{p}}$ and (2) the probability that $\widehat{\mathsf{t}}$ includes such an event is negligible. The lemma trivially follows from these two assertions.

(1) By the definition of the fail event, if such an event does not occur, then we have that all adversary events in $\widehat{\mathsf{t}}$ are valid. It then remains to show that the participant events in $\widehat{\mathsf{t}}$ are valid as well. For a given participant event of the form (P_i', L_i, m_i), we need to show that

$$\mathcal{P}(S_j, o_i, m, P_i) = (S_i, m', L_i)$$

where

1. ["deliver", k, P_i] for some k is the most recent adversary event in $\widehat{\mathsf{t}}$.

2. m is the second element in the kth adversary event in the current trace.

3. S_j is the sequence of inputs and messages received by P_i before this event in $\widehat{\mathsf{t}}$.

4. o_i is P_i's role according to $\widehat{\mathsf{t}}$.

These facts follow directly from the definition for the symbolic counterpart of simple protocols (Definition 14). By its very nature, $\mathcal{P}$ is mimicking concrete protocol p, so the only difference between protocol traces is the naming of variables – the structure of simple protocols ensures that this renaming does not affect protocol messages or outputs.

(2) Assume there exists an event of the form $[\text{“fail”}, \mathsf{m}]$. Let $M = \mathsf{m}_1, \mathsf{m}_2, ...,$ denote the messages that are communicated by Dolev Yao parties prior to the failure event, i.e. those that appear in the $[\text{“message”}, ...]$ events preceding $[\text{“fail”}, \mathsf{m}]$ in $\widehat{\mathsf{t}}$.

The failure event implies the concrete adversary created a message m that is translated to a Dolev-Yao element m not in the closure of the adversary's knowledge ($f(m) = \mathsf{m} \notin C[M]$). We show that the odds of such an event occurring are negligible.

Examine the parse tree of m. The only three operations available for creating compound messages are pairing, encryption, and signatures. By definition, membership in $C[M]$ is closed under pairing and encryption; that is, if two siblings in the parse tree are both in $C[M]$, then so is their parent. The exception is signatures – an adversary who knows a verification key and a message does not necessarily know the signature for the message under that verification key.

Consequently, if every leaf or signature in the parse tree of m has a path to the tree root with a node in $C[M]$, then $\mathsf{m} \in C[M]$. Since $\mathsf{m} \notin C[M]$, it must be the case that there is a leaf or signature m_* that has no such path node.

We thus have an adversary A that generates an m that maps to m. Using A, we construct adversary A' that will produce a bit string m_* that will map to m_*. A' simulates A to produce m then walks down the parse tree of m to m_* while recursively applying the appropriate deconstructors to m.

- If m' is a pair $\mathsf{m}_l|\mathsf{m}_r$, then A' separates $m' = \langle\text{“pair”}, m_l||m_r\rangle$ into m_l and m_r, then walks down the parse tree to the symbol containing m_* and recursively operates on the corresponding bitstring m_l or m_r.

- If m' is an encryption $\{\!|\mathsf{m}_c|\!\}_K$, then A' must decrypt $[\text{“ciphertext”}, m']$, i.e. produce what F_{CPKE} would return to the appropriate party when called with $(\mathsf{Decrypt}, m')$. Because $\mathsf{m}' \notin C[M]$, the probability A' could have independently produced m' is negligible, due to the statistical unpredictability of E. Thus, if there is a pair (m', m_c) stored in F_{CPKE}, it must be that A initiated the request $(\mathsf{Encrypt}, m_c)$ to F_{CPKE}. Because A' is simulating A it must have passed this request along, and so it knows what message m_c is the decryption of m'. On the other hand, if there is no pair (m', m_c) stored in F_{CPKE}, then the decryption of m' is $\mathsf{D}(m')$, where D was an algorithm provided by A. A' can run $\mathsf{D}(m')$ itself to obtain m_c. A' then walks down the parse tree to symbol m_c

and recursively operates on m_c.

By recursively applying the deconstruction operations, A' eventually produces a string m_* that maps to an atomic symbol or signature m_*. Notice that the only atomic symbols not in the adversary's initial view are those random nonces that were not generated by the adversary ($\mathcal{R} \setminus \mathcal{R}_{Adv}$). If m_* is a random nonce not in the closure of the adversary, then m_* was generated by an honest protocol participant and chosen uniformly from $\{0,1\}^k$, independent from the view of A'. The probability A' could generate the string m_* is 2^{-k}; since there are at most a polynomial number of k-bit strings that are valid nonces in this protocol execution, the overall probability that A' generates a string that maps to a valid nonce is $poly(k)2^{-k}$, which is negligible.

It must then be that m_* is an honest party signature. If F_{CERT} created m_* for an honest party, but $\mathsf{m}_* \notin C[M]$, then by the statistical unpredictability of S, A' can only guess the signature with negligible probability. A' couldn't have forged a new signature for an honest party's key since, under the ideal functionality, honest party signatures not generated by F_{CERT} never verify. This means the overall probability that A' produces a signature that maps to a symbolic signature outside of $C[M]$ is negligible.

Thus the probability that $\widehat{\mathsf{t}}$ includes an event of form [“fail”, m] must be negligible. $\square$

7.4 Dolev-Yao Back to UC: DY Mutual Authentication and F_{2MA}

For self-containment, we restate the mutual authentication equivalence theorem (Theorem 3) here.

Theorem 7 *Let* p *be a simple two-party protocol. Then* p *realizes* F_{2MA} *if and only if the corresponding symbolic protocol* $\widehat{\mathsf{p}}$ *satisfies Dolev-Yao 2-party mutual authentication*

Canetti and Herzog proved Theorem 7 specifically for protocols in a F_{CPKE}-hybrid, but made no use of the ideal functionality's specific nature. Their proof can be trivially extended to apply to protocols in any F-hybrid model containing multiple, composable ideal functionalities, such as the ones we have defined in this thesis. Thus, Theorem 7 concerning Dolev-Yao 2-party mutual authentication (Definition 17) and F_{2MA} (Figure 3-5) still holds for our new construction of the UCSA framework.

7.5 Using The Framework

Having stepped through and proved the different components of the universally composable symbolic analysis framework, we illustrate how it can be used by giving an informal security proof for a variation of the SPLICE authentication protocol (Figure 7-4).[2]

$$\begin{array}{ll} A \rightarrow B: & A, enc_B(A, R_A), sig_A(A, enc_B(A, R_A)) \\ A \leftarrow B: & \qquad B, A, enc_B(B, R_A) \end{array}$$

Figure 7-4: The SPLICE authentication protocol

As you can see, the SPLICE protocol has the basic form required of a simple protocol with a very natural translation to a symbolic protocol (Figure 7-5). We now show that

Let $\mathcal{D}_P$ represent the initial input/state of party P, let $*$ denote a wildcard which can be used to match anything, and let P be a place holder for the party identity each party thinks it is engaged with. We define $\mathcal{P}_{SPLICE}$ to be the mappings:

- $\{\mathcal{D}_A\} \times A \times Initiator \times \{\} \rightarrow$

  $$\{\mathcal{D}_A \cup \{R_A\}\} \times \left\langle \left\langle \left[A | \,\{\!|A|R_A|\!\}_{K_P^e} \right] \right\rangle_{K_A^v} \right\rangle \times message \times \langle Starting|A|P|K_P^e \rangle \times output$$

- $\{\mathcal{D}_A \cup \{R_A\}\} \times A \times Initiator \times \left\langle P|A|\,\{\!|P, R_A|\!\}_{K_A^e} \right\rangle \rightarrow$

  $$S^{\perp} \times \langle Finished|A|P|K_P^e \rangle \times output$$

- $\{\mathcal{D}_B\} \times B \times Responder \times \left\langle \left\langle \left[P, \{\!|P|R_P|\!\}_{K_B^e} \right] \right\rangle_{K_P^v} \right\rangle \rightarrow$

  $$S^{\perp} \times \left\langle B, P\, \{\!|B|R_P|\!\}_{K_P^e} \right\rangle \times message \times \langle Starting|B|P|K_P^e|Finished|B|P|K_P^e \rangle \times output$$

- $S^{\perp} \times * \times * \times \langle * \rangle \rightarrow$

  $$S^{*} \times \perp \times output$$

Figure 7-5: The SPLICE symbolic protocol, $\mathcal{P}_{SPLICE}$

[2]This definition is based upon a variant of the SPLICE protocol proposed by Clark and Jacob [20], as presented in [10].

$\mathcal{P}_{SPLICE}$ achieves Dolev-Yao 2-party mutual authentication (as stated in Definition 17).

If A outputs that it has finished $\mathcal{P}_{SPLICE}$ with party B, this means it must have received symbolic message $\left\langle B|A|\,\{\!|B, R_A|\!\}_{K_A^e}\right\rangle$. By the security offered by public key encryption, this could have only happened if B received (and decrypted) the symbol $\left\langle \{\!|A, R_A|\!\}_{K_B^e}\right\rangle$. Thanks to the underlying universally composable framework, if B is honest then the only way B would have produced a message containing the string R_A is if it had received the entire message $\left\langle \left\langle \left[A|\,\{\!|A|R_A|\!\}_{K_B^e}\right]\right\rangle_{K_A^v}\right\rangle$. If it had received any other message containing symbol $\left\langle \{\!|A, R_A|\!\}_{K_B^e}\right\rangle$, then that message would have been of the wrong form. Since B has received this message, B has output $\langle Finished|B|A|K_A^e\rangle$.

Similarly if B outputs that it has finished the $\mathcal{P}_{SPLICE}$ protocol with A, this means it must have received symbolic message $\left\langle \left\langle \left[A|\,\{\!|A|R_A|\!\}_{K_B^e}\right]\right\rangle_{K_A^v}\right\rangle$. If A is honest, then only A could have created this message, which implies the trace contains the event $\langle Starting|A|B|K_B^e\rangle$.

Thus both the Initiator and Responder in this protocol will only produce a successful *Finished* output if the other has produced the proper *Starting* output. This means the symbolic version of SPLICE ($\mathcal{P}_{SPLICE}$) meets the Dolev-Yao 2-party mutual authentication criteria. Now, thanks to the UCSA framework, we can further assert that the real world SPLICE protocol – implemented using a sufficiently secure key distribution system, strong signatures, and IND-CCA2 public key encryption – securely realizes ideal 2-party mutual authentication (Figure 3-5).

Chapter 8

Future Research Directions

8.1 An EUF-ACMA Realizable Ideal Signature Functionality

The previous definition of F_{SIG} [15] showed equivalence with EUF-ACMA and intuitively there does not appear to be a compelling reason why the introduction of an ideal encryption functionality should change this. The attack described in Section 4.3.3 is certainly valid, but it seems like an ideal functionality should be able to protect against such an attack without heightening the security needed to realize it. Here are two possible ways to define variants of F_{SIG} which might achieve this goal:

1. The level of abstraction used in F_{SIG} is lessened by parameterizing it with a specific signature scheme Σ in order to obtain a $\mathsf{F}_{\text{SIG}}^{\Sigma}$ which still enforces the core ideal signature functionality, but behaves in a manner consistent with Σ for extraneous cases (like handling forged signatures for messages that have existing signatures). While it may lead to more flexible secure realizations, this solution is somewhat undesirable in the UCSA framework since it would necessitate reproving the Mapping Lemma for each distinct instance of $\mathsf{F}_{\text{SIG}}^{\Sigma}$.

2. We take advantage of the fact that, in the UCSA framework, the actions of honest parties are limited to those allowed under simple protocols. This limits the power of the environment when trying to distinguish between interactions in the real and ideal worlds. In our Chapter 6 proofs, we protected against an environment which could perform any efficiently computable operation messages – it may be possible to define a

weaker ideal signature scheme ($\mathsf{F}_{\text{SIG}|\text{SP}}$) which can be securely realizes by EUF-ACMA in an environment hampered by the restrictions of simple protocols.

8.2 Other primitives and protocol goals

As this thesis shows, the UCSA framework can be extended to include protocols that use multiple cryptographic primitives. We are now limited to protocols using public key encryption and digital signatures, but as more primitives are added to the framework, the set of analyzable protocols will become richer. New protocol goals can also be added, allowing analysis of protocols with goals other than 2-party mutual authentication and key exchange.

8.3 Mixing Ideal Functionalities

Part of what makes the UC framework so powerful is the ability to abstract concrete schemes and protocols into ideal functionalities which can be used by other protocols as subroutines, secure in the knowledge that there are no potential problems arising from the composing of multiple lower-level components. There is, however, a lack of assured compatibility for different ideal functionalities used in the same protocol. In general, ideal functionalities have been formulated with the mentality that they are the sole functionality being used by a protocol or that they are being used in conjunction with other functionalities with which no conflict could arise. As Chapter 4 showed, this is not the case – ideal functionalities *can* run into problems when combined with other functionalities within a protocol. The UC and UCSA frameworks would both be greatly empowered by a methodology for assuring the compatibility of ideal functionalities within a protocol.

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). *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22, 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] Jee Hea An, Yevgeniy Dodis, Tal Rabin. On the Security of Joint Signature and Encryption. *Lecture Notes in Computer Science*, page 83. Volume 2332, Jan 2002,

[6] Michael Backes and Dennis Hofheinz. How to Break and Repair a Universally Composable Signature Functionality. Cryptology ePrint Archive, `http://eprint.iacr.org/2003/240`. 2003.

[7] Michael Backes, Brigit Pfitzmann, and Michael Waidner. Reactively Secure Signature Schemes. In *Proceedings of the 6th Information Security Conference*, 2003.

[8] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC),* pages 419–428, 1998.

[9] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Proceedings, 2004 IEEE Symposium on Security and Privacy*, pages 86–102, 2004.

[10] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

[11] Dan Boneh, editor. *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*. Springer-Verlag, August 2003.

[12] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[13] Ran Canetti. Universal composable security: A new paradigm for cryptographic protocols. In *42nd Annual Syposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, October 2001.

[14] Ran Canetti. Universally composable signatures, certification, and authentication. Cryptology ePrint Archive, `http://eprint.iacr.org/2003/239`, 2003.

[15] Ran Canetti. Universally composable signature, certification, and authentication. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop (CSFW 16)*, pages 219–233. IEEE Computer Society, June 2004.

[16] Ran Canetti and Jonathon Herzog Universally Composable Symbolic Analysis of Cryptographic Protocols (The Case of Encryption-Based Mutual Authentication and Key Exchange) Cryptology ePrint Archive, `http://eprint.iacr.org/2004/334`. 2004.

[17] Ran Canetti and Hugo Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In Birgit Pfitzmann, editor, *Advances in Cryptology - Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, May 2001.

[18] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Boneh [11], pages 565–582.

[19] Ran Canetti and Tal Rabin. Universal composition with joint state. In Boneh [11], pages 265–281.

[20] John Clark and Jeremy Jacob. On the security of recent protocols. *Information Processing Letters*, 56(3):151-155, November 1995.

[21] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal of Computing*, 30(2):391–437, 2000.

[22] D. Dolev and A. Yao. On the security of public-key protocols. *Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[23] Oded Goldreich. *Foundations of Cryptography—Volume II (Basic Applications)*. Cambridge University Press, 2004.

[24] Shafi Goldwasser, Silvio Micali, Andy Yao Strong signature schemes. *Proceedings of the 15th Annual ACM symposium on Theory of Computing (STOC '83)*, pages 431–439. ACM Press, 1983

[25] Jonathan Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, Massachusetts Institute of Technology, May 2004.

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

[27] 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.

[28] 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.

[29] 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.

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

[31] Amit Sahai Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security In *Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[32] 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.