Computationally Sound Symbolic Analysis of Probabilistic Protocols with Ideal Setups [⋆]

Zhengqin Luo

INRIA Sophia-Antipolis

Abstract. Recently, many approaches have been proposed for building simple symbolic proofs of cryptographic protocols with computational soundness. However, most of them support only bare-bone execution model without any ideal setup, such as the existence of authenticated channel, and only deterministic protocols. Thus many protocols are not expressible in those models. Following the work of Canetti and Herzog [1], we propose a probabilistic symbolic model for analyzing cryptographic protocols and a general way of incorporating ideal setups by using a probabilistic process calculus. Each ideal setup in the symbolic model will correspond to an ideal functionality in the computational model. Furthermore, we show the computational faithfulness of this symbolic model with respect to a hybrid computational model in which ideal functionalities are employed.

1 Introduction

Proving whether cryptographic protocols are secure is one of the central problems in modern cryptography. Over the last two decades, there are mainly two views on analyzing these protocols. On the one hand, although the computational approach provides a rigorous framework for defining and proving security properties, proofs often involve tedious reductions from probabilistic algorithms to underlying cryptographic schemes, which are generally hard to be automated. On the other hand, the formal method approach proposes a much simpler model for describing and analyzing protocols by using abstract term algebra for modeling perfect cryptography, opening doors to numerous automated tools (or partially automated) in this area. However, the adversary's behavior is restricted by a pre-defined set of operations, which might not capture all possible attacks in the computational model.

Recently, researchers have been making efforts on linking these two approaches, enjoying the simplicity of the formal approach and the rigor in the computational approach at the same time. We confine us now on the method of obtaining soundness results of Dolev-Yao [2] style model with respect to the computational model. The seminal work is proposed by Abadi and Rogaway in [3]. They propose a simple language of encryption terms, and show that equivalence

[⋆] This work was partially done when the author was in Shanghai Jiao Tong University.

between symbolic terms implies computational indistinguishability between ensembles generated by replacing formal encryptions with concrete schemes. Their work only addresses the case of symmetric encryption in the presence of a passive adversary. Subsequently, it is significantly extended to deal with public-key encryption scheme, signature scheme even in the presence of an active adversary [4–10].

Previous works focus on soundness results of symbolic abstraction of encryption or signature in deterministic protocol models. However, a large number of protocols utilize randomization at their behavioral level, or employ basic cryptographic protocols as building blocks, such as coin-tossing protocol, or even assume the existence of authenticated channel or anonymous channel.

In this work, we consider to introduce *probability* into the symbolic model to address the expressiveness of internal probabilistic behaviors in protocols. And then we consider to introduce *ideal setups* for modeling network assumptions and simple two-party protocols as cryptographic building blocks. The notion of ideal setup is inspired by the ideal functionality in the Universally Composable (UC) security framework [11], a computational framework for designing and analyzing protocols. In that framework, ideal functionality can be used to characterize network assumptions as well as specify security requirements of protocols. Similarly, the ideal setups in our framework play two important roles:

– Modeling network assumptions, such as authenticated channel, anonymous channel, and etc.
– Modeling cryptographic protocols as basic building blocks, such as coin-tossing protocol, oblivious transfer protocol, and zero-knowledge proof protocol, for constructing large and complex protocols.

Ideal setups should capture appropriate intuitions for each tasks. It provides a modular approach for analyzing symbolic protocols: when analyzing a protocol which invokes basic cryptographic protocols mentioned above, we can analyze only an "abstract" protocol using corresponding ideal setups, and then claim that the original protocol satisfies same properties as the "abstract" protocol. We follow the approach of Canetti and Herzog [1], using the UC framework as the underlying computational framework.

The contributions of our paper are:

– We define a simple language for describing protocol programs which can be both interpreted in the symbolic model and the computational model. By the language, we are able to express multi-party protocols in which parties can make internal probabilistic choice, employ public-key encryption and signature, or use ideal setups. The language supports two ideal setups: authenticated channel and coin-tossing protocol (Section 3).
– We show how to interpret protocol programs in our symbolic model by using a subset of the Probabilistic Applied Pi (PAPi) calculus [12]. Each ideal setup is implemented by an auxiliary process (Section 4).
– We also explain how to translate protocol programs into a certain hybrid model of the UC framework, in which ideal functionalities corresponding to ideal setups in the symbolic model (Section 5).

– We finally demonstrate the faithfulness of our symbolic model with respect to the hybrid model. That is, almost all attacks in the hybrid model can be interpreted in the symbolic model, except for negligible probability (Section 6).

Related works The efforts on linking the symbolic approach and the computational one, especially showing soundness results of Dolev-Yao model with respect to certain computational model, were initiated by Abadi and Rogaway [3], and were extended by [4–7] in several aspects. Backes, Pfitzmann and Waidner proposed an abstract cryptographic library based on the reactive simutability setting [13–15]. Their symbolic model supports nested operation of cryptographic primitives, such as symmetric and public-key encryption, signature, and message authenticated code under arbitrary active adversary. They also demonstrated for several protocols that Dolev-Yao style proof implies the computational security [16–18]. Based on the UC framework, Canetti and Herzog established the soundness of symbolic analysis with respect to the UC framework [1]. However, they only considered a restricted class of protocols and only support certified public-key encryption. Patil extended it to handle also standard signature [19].

2 Background

This section provides necessary background information on the topic. We first introduce a subset of the PAPi calculus. We then briefly review the UC framework and the Universally Composable Symbolic Analysis (UCSA) framework

2.1 A Subset of PAPi calculus

The applied pi calculus is an extension of the pure pi calculus. It introduces terms and equations over terms for modeling cryptographic primitives, and uses techniques in process algebra to reason about distributed systems and protocols. The PAPi extends it into a probabilistic framework, allowing analysis of probabilistic processes.

The basics elements in the PAPi calculus are a set of *names*, a set *variables*, and a *signature* Σ which consist of a finite set of function symbols with arities. Terms can be defined by applying function symbols in Σ on names, variables and terms. We can equip a given signature Σ with an *equational theory* E to relate two terms which are syntactically different but semantically equivalent.

The grammar of plain processes and extended processes are given below:

$$\begin{aligned} P, Q \ ::= \mathbf{0} \ \big| \ \bar{u}\langle M\rangle.P \ \big| \ u(x).P \ \big| \ P \oplus_{\frac{1}{2}} Q \ \big| \ !P \ \big| \\ P|Q \ \big| \ \nu n.P \ \big| \ \text{if } M = N \text{ then } P \text{ else } Q \end{aligned}$$

$$A, B \ ::= P \ \big| \ \nu n.A \ \big| \ \nu x.A \ \big| \ A|B \ \big| \ \{M/x\}$$

Comparing to the original grammar of plain process in the PAPi calculus, we omit the non-deterministic choice operator, and confine the probabilistic choice operator $\oplus_p$ to only one half probabilistic operator $\oplus_{\frac{1}{2}}$. Extending the plain processes with active substitution $\{M/x\}$, we are able to reason about the static knowledge exposed by a certain process. An evaluation context $C[\_]$ is a process with a hole under restriction or parallel composition. For space reason, we do not present the semantics of PAPi here, it can be found in [20, 12, 21]

2.2 The UC framework and UCSA framework

The UC framework provides a general methodology for designing and analyzing cryptographic protocols, especially asserting whether a given protocol securely realizes its security specification. The most salient feature of this framework is the strong composable property, by which one can ensure that a protocol still maintains its security properties when being executed in an arbitrary unpredictable environment, or being composed in a modular way. An comprehensive overview of the UC framework can be found in [11].

The UCSA framework facilitates the *universal composition theorem* to simplify their framework of sound symbolic analysis. The symbolic Dolev-Yao model is a simplified model for analyzing two-party deterministic protocol which uses only public-key encryption. Each protocol peer is defined by a mapping from current state and incoming messages to outgoing messages. The adversary is only limited to a set of symbolic operations according to the rules representing its limitation with respect to perfect cryptography.

Instead of establishing the computational faithfulness of the symbolic model with respect to a concrete computational model using concrete public-key encryption scheme, the UCSA shows that the symbolic model is faithful for a hybrid model in the UC framework. The hybrid model uses $\mathcal{F}_{\text{CPKE}}$ for idealized encryption service which is secure unconditionally, even in the presence of a computational unbounded adversary. Since $\mathcal{F}_{\text{CPKE}}$ can be UC-realized by any CCA-secure[1] encryption scheme, it serves as a "bridge" between the symbolic model and the concrete model: when we obtain a sound symbolic proof of protocol in the hybrid model, we can facilitate the UC theorem to replace each instance of $\mathcal{F}_{\text{CPKE}}$ to an instance of CCA-secure encryption scheme, while maintaining the security in a computational sense at the same time.

3 A Simple Language for Probabilistic Protocols

We first present a simple language for describing behavior of probabilistic protocols. The language can be used to describe high-level codes of protocols without any implementation detail of network communication and underlying cryptography. We could compile a protocol program either into symbolic model by using abstract term algebra, or into computational model by using concrete cryptographic schemes.

[1] Chosen Ciphertext Attack

Definition 1 (Protocol Program). *Fixed a finite set of identifier of parties* $\mathcal{C} = \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{M}\}$*, a protocol program is defined by a function* $\mathcal{P} = \{(\mathtt{A}, \mathtt{P}_\mathtt{A}), (\mathtt{B}, \mathtt{P}_\mathtt{B}), \ldots, (\mathtt{M}, \mathtt{P}_\mathtt{M})\}$*, mapping each identifer to a program defined by the grammar below, where* x, m, c, s, b *represent variables for different types of values.*

$$
\begin{aligned}
R &::= \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{M} \\
B &::= \mathtt{true} \mid \mathtt{false} \\
I &::= x := \mathtt{newnonce}() \mid x := \mathtt{encrypt}(m, R) \mid x := \mathtt{decrypt}(c) \mid \\
&\quad\; x := \mathtt{sign}(m) \mid x := \mathtt{verify}(m, s, R) \mid x := \mathtt{pair}(m_1, m_2) \mid \\
&\quad\; x := \mathtt{fst}(m) \mid x := \mathtt{snd}(m) \mid x := R \mid x := B \\
E &::= \mathtt{input}(x) \mid \mathtt{output}(m) \mid \mathtt{send}(m) \mid \mathtt{recv}(x) \mid \\
&\quad\; \mathtt{send}^\mathtt{a}(m, R) \mid \mathtt{recv}^\mathtt{a}(x, R) \mid \mathtt{coin}^\mathtt{i}(b, R) \mid \mathtt{coin}^\mathtt{r}(b, R) \\
P &::= I \mid E \mid P; P \mid \mathtt{if}\ x = y\ \mathtt{then}\ P\ \mathtt{else}\ P \mid \mathtt{prob}(P; P)
\end{aligned}
$$

Internal Computations The grammar structure I defines atomic internal computation for protocol parties. A party could generate a fresh random nonce by command `newnonce`. It could also encrypt a message with someone's public-key by `encrypt`, or decrypt a ciphertext with its own private-key by `decrypt`. We assume that each party's identity is bound to its public-key, which cannot be revealed to others. Thus, we use an identity of a party instead of a public-key when encrypting. The commands for generating and verifying signatures, `sign` and `verify`, are similar to the case of public-key encryption. `pair`, `fst` and `snd` are standard paring operation.

External Interactions `input` and `output` are used to obtain input from the environment and return output to it. `send` and `recv` model the sending and receiving over an adversary-controlled network. Intuitively, it models an asynchronous network, and the adversary could learn, intercept, modify, and re-schedule messages over this network.

Ideal Setups In addition to common external interactions, we introduce two ideal setups. The first one is the ideal setup of authenticated channel. A party could send an authenticated message m to party B by command $\mathtt{send}^\mathtt{a}(m, \mathtt{B})$, and could wait to receive an authenticated message from party B by $\mathtt{recv}^\mathtt{a}(x, \mathtt{B})$ and store it in x. By our assumption, the adversary should not be able to modify or reproduce authenticated messages, but it can learn and intercept them. The second one is the ideal setup of coin-tossing protocol. Commands $\mathtt{coin}^\mathtt{i}$ and $\mathtt{coin}^\mathtt{r}$ allow two parties to initiate an ideal coin-tossing protocol as initiator and responder, respectively. By assumptions, the adversary should not be able to influence the fairness of the common coin by any means.

Control Flows We provide three types of control flows here. The sequential execution $(\mathtt{P}_1; \mathtt{P}_2)$ and conditional execution $(\mathtt{if}\ M = N\ \mathtt{then}\ \mathtt{P}_1\ \mathtt{else}\ \mathtt{P}_2)$ are usual. Command $\mathtt{prob}(\mathtt{P}_1, \mathtt{P}_2)$ means executing $\mathtt{P}_1$ with probability 0.5 and executing $\mathtt{P}_2$ with the rest. The reason why we only model one half choice here is that choices with arbitrary probability could always be approximated by multiple one half choices.

For example, a simple challenge-response protocol can be described as follows:

```
A's program :                          B's program :

$N_a$ = newnonce();                    recv($m'_1$);
$x_1$ = pair($N_a$, B);                $x'_1$ = decrypt($m'_1$);
$m_1$ = encrypt($x_1$, B);             $N$ = fst($x'_1$);
send($m_1$);                           send($N$);
recv($m_2$);
if $m_2 = N_a$ then $output$(true);
```

Fig. 1. Protocol program for a simple challenge-response protocol

4 Symbolic Interpretation

In this section, we show how to translate a protocol program into a symbolic protocol in our symbolic model which is described by the subset of PAPi calculus introduced in Section 2.1. The abstract term algebra is modeled by an equational theory. Each single party is modeled by a process in the calculus. Also, we demonstrate how to characterize ideal setups by auxiliary processes.

Equational theory We use two types of function symbol for different purposes, as showed below.

- Constant: `true`/0, `false`/0, `garb`/0.
- Cryptographic operator: `enc`/2, `dec`/2, `pk`/1, `sign`/2, `ver`/3, `vk`/1, `pair`/2, `fst`/1, `snd`/1.

The number followed by each function symbol indicates its arity. The `true` and `false` are boolean constants. `grab` refers to ill-formed terms such as an inappropriate decrypted term. The operations of encryption, signature and paring are defined in a usual way. The cryptography is modeled in a Dolev-Yao style as being perfect. The equations are given in Figure 2. These equations are fairly

$$\begin{aligned}
&\mathtt{fst}(\mathtt{pair}(x, y)) = x\\
&\mathtt{snd}(\mathtt{pair}(x, y)) = y\\
&\mathtt{dec}(\mathtt{enc}(x, \mathtt{pk}(y)), y) = x\\
&\mathtt{ver}(x, \mathtt{sign}(x, y), \mathtt{vk}(y)) = \mathtt{true}
\end{aligned}$$

Fig. 2. Equational theory E for symbolic protocols

standard for those primitives. The function symbol `pk` maps user's private key to its public key. One can use `dec` and its private key to decrypt a message encrypted under corresponding public key. Without the private key, the adversary cannot learn anything about the message. The case for signature is similar. In addition, we require that improperly operated terms equate to `garb`, such as decryption of a non-encrypted term.

Translating protocol program Given a protocol program $\mathcal{P}$ for identity set $\mathcal{C}$. For every $\mathtt{R} \in \mathcal{C}$, we translate $\mathcal{P}(\mathtt{R})$ to a process $\mathcal{S}_\mathtt{R}$ which uses the following channel names to interact with other processes.

- $input_\mathtt{R}$, $output_\mathtt{R}$: Obtaining inputs and returning outputs;
- $send_\mathtt{R}$, $recv_\mathtt{R}$: Sending and receiving over the adversary-controlled network;
- $send^a_\mathtt{RA}$, $recv^a_\mathtt{AR}$: Sending and receiving authenticated messages with $\mathtt{A}$, for $\mathtt{A} \in \mathcal{C}$;
- $toss^i_\mathtt{RA}$, $toss^r_\mathtt{AR}$: Initiating and responding coin-tossing with $\mathtt{A}$, for $\mathtt{A} \in \mathcal{C}$.

First, internal computations in $\mathcal{P}(\mathtt{R})$ will be translated to operations of function symbols according to the equational theory E. Then, external interactions will be translated to interactions on channels defined above. Finally, control flows in $\mathcal{P}(\mathtt{R})$ are expressible in PAPi calculus.

Definition 2 (Symbolic interpretation of protocol program). *Given an identity $\mathtt{R} \in \mathcal{C}$, a single protocol program $\mathcal{P}(\mathtt{R})$ can be inductively translated to a process $S_\mathtt{R}$ by rules defined in Appendix A.*

In Appendix A, We also give the translation of the simple challenge-response protocol in Figure 1.

Modeling ideal setups We construct auxiliary processes to capture the intuition of our ideal setups of authenticated channel and coin-tossing protocols.

When channels are authenticated, if $\mathtt{B}$ receives a message m from $\mathtt{A}$, $\mathtt{A}$ must have sent m before. Furthermore, if $\mathtt{A}$ sent m to $\mathtt{B}$ only k times, then $\mathtt{B}$ will not receive m more than k times. Notice that the secrecy of those transmitted messages is not guaranteed, and messages are transmitted asynchronously. We use the following process to help $\mathtt{A}$ and $\mathtt{B}$ to exchange a message over channel $send^a_\mathtt{AB}$ and $recv^a_\mathtt{AB}$.

$$\begin{aligned}\mathcal{A}^a_s(\mathtt{A},\mathtt{B}) ::= {} & send^a_\mathtt{AB}(x). \\ & \nu c.(\overline{adv^a_\mathtt{AB}}\langle (x,c)\rangle. \\ & c(y). \\ & if\ y = x\ then\ \overline{recv^a_\mathtt{AB}}\langle x\rangle)\end{aligned}$$

The channels $send^a_\mathtt{AB}$ and $recv^a_\mathtt{AB}$ will be restricted between parties and the auxiliary process only, which are invisible to the adversary. Thus, the adversary cannot modify or reproduce a message. However, it can learn the message through channel $adv^a_\mathtt{AB}$. The fresh channel name c is used to distinguish different instances of the auxiliary processes between the same two participants. We use replication operator to allow each participant in set $\mathcal{C}$ being able to send and receive unlimited number of messages.

$$\mathcal{A}^a ::= \prod_{\mathtt{A},\mathtt{B}\in\mathcal{C},\mathtt{A}\neq\mathtt{B}} !\mathcal{A}^a_s(\mathtt{A},\mathtt{B})$$

For the ideal setup of coin-tossing protocol, it requires that two parties should be able to generate an unbiased random bit value. The adversary should not be able to influence the result by any means. The auxiliary processes for generating a single bit can be defined as follows.

$$\begin{aligned}\mathcal{A}^t_{half}(\mathtt{A},\mathtt{B},x) ::= {} & \nu c.\ \Big(toss^i_{\mathtt{AB}}().\overline{adv^i_{\mathtt{AB}}}\langle c\rangle.c().\overline{toss^i_{\mathtt{AB}}}\langle x\rangle\Big) \mid \\ & \nu c.\ \big(toss^r_{\mathtt{AB}}().\overline{adv^r_{\mathtt{AB}}}\langle c\rangle.c().\overline{toss^r_{\mathtt{AB}}}\langle x\rangle\big)\end{aligned}$$

$$\mathcal{A}^t_s(\mathtt{A},\mathtt{B}) ::= \mathcal{A}^t_{half}(\mathtt{A},\mathtt{B},\mathtt{true}) \oplus_{0.5} \mathcal{A}^t_{half}(\mathtt{A},\mathtt{B},\mathtt{false})$$

The participant $\mathtt{A}$ initiates an instance of the ideal protocol via the channel name $toss^i_{\mathtt{AB}}$. Upon the request of the adversary, $\mathtt{A}$ receives an unbiased random bit. Then, the participant $\mathtt{B}$ as a responder gets the same random bit via channel $toss^r_{\mathtt{AB}}$ upon the request of the adversary. It can be seen that the choice of the random bit is independent to the behavior of the adversary. By using replication, we obtain an ideal process enabling each pair of parties to generate arbitrary long random strings.

$$\mathcal{A}^t ::= \prod_{\mathtt{A},\mathtt{B}\in\mathcal{C},\mathtt{A}\neq\mathtt{B}} !\mathcal{A}^t_s(\mathtt{A},\mathtt{B})$$

Adversary and executions Given all the processes of the participants in the protocol, the process denoting whole protocol can be defined as follows, where $\mathcal{V}$ contains channel name between parties and auxiliary processes, which should not be observed by the adversary:

$$\mathcal{S} = \nu\mathcal{V}.\ (\ \mathcal{S}_{\mathtt{A}} \mid \mathcal{S}_{\mathtt{B}} \mid \ \ldots \ \mid \mathcal{S}_{\mathtt{M}} \mid \mathcal{A}^a \mid \mathcal{A}^i)$$

The adversary could freely interact with $\mathcal{S}$ by the following free channel names:

– $send_{\mathtt{A}}$, $recv_{\mathtt{A}}$: Controlling the insecure network;
– $adv^a_{\mathtt{AB}}$: Influence the auxiliary process of an authenticated channel;
– $adv^i_{\mathtt{AB}}$, $adv^r_{\mathtt{BA}}$: Influence the auxiliary process of an instance of coin-tossing protocol;

Naturally, the adversary can be modeled as a process $\mathcal{P}_{adv}$, which only contains free channel name described above. Also, we could construct an environment process $\mathcal{P}_{env}$ that provides input to each user process $\mathcal{S}_{\mathtt{R}}$ and receives outputs from them. We require that $\mathcal{P}_{env}$ contains only channel name $input_{\mathtt{R}}$ and $output_{\mathtt{R}}$, for $\mathtt{R} \in \mathcal{C}$. Given the adversary process $\mathcal{P}_{adv}$, an environment process $\mathcal{P}_{env}$, we can construct a context $C^{env}_{adv}[\_] = (\mathcal{P}_{adv}|\mathcal{P}_{env}|\_)$. Thus the execution of the protocol in the presence of the adversary can be seen as the interaction between $\mathcal{S}$ and $C^{env}_{adv}[\_]$.

5 Computational (Hybrid) Interpretation

Next, we define the computational model in our framework. It is a hybrid model in the UC framework which supports ideal functionalities. We elaborate ideal functionalities for our ideal setups, and then present how to interpret protocol programs in this model. Notice that we do not directly translate symbolic protocols to concrete protocols, avoiding implementation details in the symbolic models.

Ideal functionalities for encryption and signature Recall that public-key encryption scheme and signature scheme denote cryptographic primitives which can be implemented by local algorithms. Our formulation of these operations is the same as the original UCSA framework [1, 19]. The public-key encryption is modeled by the certified public-key encryption functionality $\mathcal{F}_{\text{CPKE}}$, and the digital signature is modeled by the certification functionality $\mathcal{F}_{\text{CERT}}$. As we have assumed, these functionalities do not deal with key issues, and provide unconditional security, in which ciphertext (or signature) bears no computational relation to plaintext. Each party uses identities during encrypting/decrypting and signing/verifying. Note that $\mathcal{F}_{\text{CPKE}}$ can be securely realized by a CCA-secure encryption scheme with a trusted key-registration service. Similarly, $\mathcal{F}_{\text{CERT}}$ can be securely realized by a CMA-secure[2]. We do not give the detail definitions here, interested readers are refer to [1, 19].

Ideal functionalities for ideal setups For the ideal setup of authenticated channel, we present the authenticated communication ideal functionality $\mathcal{F}_{\text{AUTH}}$, as shown in Figure 3. It is originally formulated by Canetti in [11]. We omit the case

Functionality $\mathcal{F}_{\text{AUTH}}$

1. Upon receiving an input (Send, sid, m) from party P, do: If $sid = (P, R, sid')$ for some R, then generate a public delayed output (Send, sid, m) to R and halt. Otherwise ignore the input.

Fig. 3. Authenticated Communication Ideal functionality

of corrupted parties, since we do not consider party corruption in our symbolic model. One could see that the behavior of an instance of $\mathcal{F}_{\text{AUTH}}$ between $\mathtt{A}$ and $\mathtt{B}$ is essentially identical to the ideal process $\mathcal{A}^a_s(\mathtt{A}, \mathtt{B})$.

For the ideal setup of coin-tossing protocol. We present the coin-tossing ideal functionality $\mathcal{F}_{\text{CT}}$, as shown in Figure 4. This functionality can be thought as a "bridge" between a concrete cryptographic protocol of coin-tossing and the ideal

[2] Chosen Message Attack

Functionality $\mathcal{F}_{\mathrm{CT}}$

$\mathcal{F}_{\mathrm{CT}}$ proceeds as follows, running with parties P and R.

1. Upon receiving an input $(\mathsf{Toss}, sid, P, R)$ from party P, randomly choose a bit value r, and send $(\mathsf{Tossed}, sid, P)$ to the adversary S.
2. Upon receiving $(\mathsf{Result}, sid, P)$, send $(\mathsf{Tossed}, sid, r)$ to P.
3. Upon receiving an input $(\mathsf{Toss}, sid, P, R)$ from party R, send $(\mathsf{Tossed}, sid, R)$ to the adversary S.
4. Upon receiving $(\mathsf{Result}, sid, R)$, send $(\mathsf{Tossed}, sid, r)$ to R.

Fig. 4. Coin-tossing Ideal Functionality

process of coin tossing in our symbolic model. We can see that the coin is generated independently by $\mathcal{F}_{\mathrm{CT}}$, which is certainly uncorrelated to the adversary's behavior.

Executing protocol program Given a protocol program $\mathcal{P}$ with a set of identity $\mathcal{C}$, we specify an Interactive Turing Machine to interpret for a single protocol program $\mathcal{P}(\mathtt{R})$ for each $\mathtt{R} \in \mathcal{C}$.

We briefly sketch how this machine performs. Given a single protocol program $\mathcal{P}(\mathtt{R})$, ITM $M_{\mathtt{R}}$ maintains a memory state Σ which maps each variable occurred in $\mathcal{P}(\mathtt{R})$ into a concrete bit-string, with $\mathtt{PID}_{\mathtt{R}}$ as its concrete party identifier. Each bit-string is tagged with its type in order to maintain a structure of each message as in the abstract term algebra. For example, a bit-string c of ciphertext will be recorded as $\langle\text{"ciphertext"}, c\rangle$, and bit-string r of random nonce will be recorded as $\langle\text{"nonce"}, r\rangle$. At the beginning, each variable is mapped to an uninitiated value. Then $M_{\mathtt{R}}$ starts to interpret commands $\mathcal{P}(\mathtt{R})$ one by one, updating the memory state. Commands related to cryptographic operations and ideal setups will be executed as calls to appropriate instances of ideal functionalities.

- $x = \mathtt{encrypt}(m, \mathtt{A})$: Send $(\mathtt{Encrypt}, \mathtt{PID}_{\mathtt{A}}, m)$ to $\mathcal{F}_{\mathrm{CPKE}}$, receive c, and update x with $\langle\text{"ciphertext"}, c\rangle$.
- $x = \mathtt{decrypt}(c)$: If the value of c is $\langle\text{"ciphertext"}, c'\rangle$, then send $(\mathtt{Decrypt}, \mathtt{PID}_{\mathtt{R}}, c')$ to $\mathcal{F}_{\mathrm{CPKE}}$, receive m, and update x with m.
- $x = \mathtt{sign}(m)$: Send $(\mathtt{Sign}, \mathtt{PID}_{\mathtt{R}}, m)$ to $\mathcal{F}_{\mathrm{CERT}}$, receive s, and update x with $\langle\text{"signature"}, s\rangle$.
- $x = \mathtt{verify}(m, s, \mathtt{A})$: If the value of s is $\langle\text{"signature"}, s'\rangle$, then send $(\mathtt{Verify}, \mathtt{PID}_{\mathtt{A}}, m, s')$ to $\mathcal{F}_{\mathrm{CERT}}$, receive b, and update x with $\langle\text{"boolean"}, b\rangle$.
- $\mathtt{send}^{\mathtt{R}}(m, \mathtt{A})$: Send $(\mathtt{Send}, \langle\mathtt{R}, \mathtt{A}\rangle, m)$ to $\mathcal{F}_{\mathrm{AUTH}}$.
- $\mathtt{recv}^{\mathtt{R}}(x, \mathtt{A})$: Receive m from $\mathcal{F}_{\mathrm{AUTH}}$, and update x with m.
- $\mathtt{coin}^{\mathtt{i}}(b, \mathtt{A})$: Send $(\mathtt{Toss}, \langle\mathtt{R}, \mathtt{A}\rangle)$ to $\mathcal{F}_{\mathrm{CT}}$, receive b, and update x with $\langle\text{"boolean"}, b\rangle$
- $\mathtt{coin}^{\mathtt{r}}(b, \mathtt{A})$: Send $(\mathtt{Toss}, \langle\mathtt{A}, \mathtt{R}\rangle)$ to $\mathcal{F}_{\mathrm{CT}}$, receive b, and update x with $\langle\text{"boolean"}, b\rangle$

Particularly, when it encounters a $\mathtt{prob}(\mathtt{P}_1, \mathtt{P}_2)$ command, it flips a coin and decides which branch to follow. Rest of these commands are executed in a standard way. Thus, we could obtain a set of ITMs $\mathcal{P}_h = \{M_{\mathtt{A}}, M_{\mathtt{B}}, \ldots, M_{\mathtt{M}}\}$ for executing the protocol program in the hybrid model.

6 Faithfulness of the Symbolic Model

In this section, we first define execution traces of symbolic protocol and hybrid protocol, and then we show our symbolic model is faithful with respect to our hybrid model.

Definition 3 (Trace of symbolic protocol). *Let $\mathcal{P}_s = \{\mathcal{S}_{\mathtt{A}}, \mathcal{S}_{\mathtt{B}}, \ldots, \mathcal{S}_{\mathtt{M}}\}$ be a symbolic protocol. Given the adversary process P_{adv} and the environment process P_{env}, the execution of the protocol can be viewed as interaction between the process $\mathcal{S} = \nu\mathcal{V}.\big(\ \mathcal{S}_{\mathtt{A}} \mid \mathcal{S}_{\mathtt{B}} \mid \ \ldots \ \mid \mathcal{S}_{\mathtt{M}} \mid \mathcal{A}^a \mid \mathcal{A}^i\big)$ and the context $C^{env}_{adv}[\_] = (\mathcal{P}_{adv}|\mathcal{P}_{env}|\_)$ under a scheduler F for resolving non-determinism. Given a terminating execution $e = \mathcal{S} \xrightarrow{\alpha_1}_{\mu_1} \mathcal{S}_2 \xrightarrow{\alpha_2}_{\mu_2} \cdots \xrightarrow{\alpha_k}_{\mu_k} \mathcal{S}_k$, the symbolic trace $\mathtt{tr}(e)$ is defined as: $\mathtt{tr}(e) = \mathtt{tr}(\alpha_1); \mathtt{tr}(\alpha_2); \ldots; \mathtt{tr}(\alpha_k)$. The definition of $\mathtt{tr}$ is given in Appendix B*[3].
Let $\mathrm{STRACE}^{F,\mathcal{P}_{adv}}_{\mathcal{P}_s,\mathcal{P}_{env}}$ be the random variable of symbolic traces, such that for every terminated execution e:

$$\Pr[\mathrm{STRACE}^{F,\mathcal{P}_{adv}}_{\mathcal{P}_s,\mathcal{P}_{env}} = \mathtt{tr}(e)] = Prob^F_{\mathcal{S}}(e)$$

We show that the adversary's power is limited by the equational theory in our symbolic model.

Proposition 1 (Closure property). *Given a symbolic trace t, such that*

$$\Pr[\mathrm{STRACE}^{F,\mathcal{P}_{adv}}_{\mathcal{P}_s,\mathcal{P}_{env}} = t] > 0$$

Let t_i in t be an adversary event with term M. Let ϕ be the frame recovered from $t_1, \ldots, t_{i-1}$. Then we have

$$\phi \vdash M$$

Proof. It is straightforward since the adversary $\mathcal{P}_{adv}$ is also a legal PAPi process. $\square$

For the execution trace of hybrid protocols, we record activations of each entity in a sequential manner.

Definition 4 (Traces of hybrid protocols). *Let $\mathcal{P} = \{M_{\mathtt{A}}, M_{\mathtt{B}}, \ldots, M_{\mathtt{M}}\}$ be a hybrid protocol. Given the environment $\mathcal{Z}$, the input z, a security parameter k. Let r_c be random inputs for $\mathcal{Z}$, $\mathcal{F}_{\mathrm{CPKE}}$, and $\mathcal{F}_{\mathrm{CERT}}$, r_b be random inputs for each party and $\mathcal{F}_{CT}$. The execution trace $\mathrm{TRACE}_{\mathcal{P},\mathcal{Z}}(z, k, r_c, r_b)$ can be inductively defined by rules given in Appendix C.*
Let $\mathrm{TRACE}_{\mathcal{P}_h,\mathcal{Z}}(k, z)^*$ be the random variable describing $\mathrm{TRACE}_{\mathcal{P}_h,\mathcal{Z}}(k, z, r_c, r_b)$ when r_c and r_b are uniformly chosen.

[3] The execution trace and its probability are defined in [12]

Then, we define a mapping from hybrid traces to symbolic traces, translating each concrete event in hybrid traces into a symbolic event. Furthermore, we also define the validity of translated trace.

Definition 5 (Mapping from hybrid trace to symbolic trace). *Let t be a hybrid trace of an execution of hybrid protocol $\mathcal{P}_h$. We inductively define the mapping of t to a symbolic trace $symb(t)$ in two steps:*

1. *First, we translate each concrete string in each hybrid event t_i to corresponding symbolic terms, inductively by defining a partial mapping f from bit-strings to symbolic terms. Recall that we use a type tag for different types of values, this could help us to reconstruct symbolic terms using function symbols. We map all ill-formed bit-string to garbage term* `garb`*.*
2. *Secondly, we convert each hybrid event $\mathtt{E}_i$ to a symbolic interaction t_i, replacing concrete string in $\mathtt{E}_i$ into corresponding symbolic terms using f. If t_i is an adversary event with term M which does not satisfy closure property, then set $t_i = [\text{“}fail\text{”}, M]$*

If t is a random variable of hybrid traces, then $symb(t)$ would be random variable of symbolic traces.

Definition 6 (Valid symbolic traces). *Given a symbolic protocol $\mathcal{P}_s$ and a symbolic traces $symb(t)$ generated from hybrid trace t. We say that t is valid for $\mathcal{P}_s$ if and only if there exist an adversary process $\mathcal{P}_{adv}$, an environment processes $\mathcal{P}_{enc}$, and a scheduler F, such that*

$$\Pr[\mathrm{STRACE}^{F,\mathcal{P}_{adv}}_{\mathcal{P}_s,\mathcal{P}_{env}} = symb(t)] > 0$$

Otherwise, we say that t is not valid for $\mathcal{P}_s$.

Now, we are ready to state the faithfulness theorem of our symbolic model. Intuitively, the theorem says that all but a negligible fraction of hybrid traces can be interpreted in our symbolic model. Therefore, the limited symbolic adversary precisely captures the ability of a more powerful adversary in the hybrid model. Relying on this, it is suffice to analyze protocols in the symbolic model, and then claim that its hybrid counterpart satisfies same properties as the symbolic protocol.

Theorem 1 (Faithfulness of the symbolic model). *For all protocol program $\mathcal{P}$, environments $\mathcal{Z}$, and inputs z of length polynomial in the security parameter k,*

$$\Pr[t \leftarrow \mathrm{TRACE}_{\mathcal{P}_h,\mathcal{Z}}(k,z)^* : symb(t) \textit{ is not valid for } \mathcal{P}_s] \leq \epsilon(k)$$

where ϵ is negligible[4]*.*

[4] ϵ is *negligible* if $\forall c \geq 0.\exists k_c$ s.t. $\forall k \geq k_c.\epsilon(k) \leq k^{-c}$

7 Conclusions

In this paper, we have presented a probabilistic framework for computationally sound symbolic analysis of security protocols. We introduced ideal setups for modeling network assumptions as well as two-party cryptographic primitives, and we then showed how to interpret them in both symbolic model and hybrid model. Finally we demonstrated that our symbolic model is faithful with respect to the hybrid model.

For future research directions, one promising direction is to introduce more ideal setups to the symbolic model, such as anonymous channel, blind signature scheme, commitment protocol, or even zero-knowledge proof protocol. Another interesting direction is to develop automated verification technique for the probabilistic symbolic model to obtain sound automated proof of cryptographic protocols.

Acknowledgment The author would like to thank Xiaojuan Cai for helpful discussion while evaluating this work. The author also thank Jun Pang and Tamara Rezk for insightful comments. Finally, many thanks to the anonymous referees for helpful feedback.

References

1. Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In: TCC. (2006) 380–403
2. Dolev, D., Yao, A.C.C.: On the security of public key protocols (extended abstract). In: FOCS. (1981) 350–357
3. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). In: IFIP TCS. (2000) 3–22
4. Abadi, M., Jürjens, J.: Formal eavesdropping and its computational interpretation. In: TACS. (2001) 82–94
5. Horvitz, O., Gligor, V.D.: Weak key authenticity and the computational completeness of formal encryption. In: CRYPTO. (2003) 530–547
6. Micciancio, D., Warinschi, B.: Completeness theorems for the Abadi-Rogaway language of encrypted expressions. Journal of Computer Security **12**(1) (2004) 99–130
7. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: TCC. (2004) 133–151
8. Janvier, R., Lakhnech, Y., Mazaré, L.: Completing the picture: Soundness of formal encryption in the presence of active adversaries. In: ESOP. (2005) 172–185
9. Kremer, S., Mazaré, L.: Adaptive soundness of static equivalence. In: ESORICS. (2007) 610–625
10. Comon-Lundh, H., Cortier, V.: Computational soundness of observational equivalence. Technical report, INRIA (2008)
11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. (2001) 136–145
12. Goubault-Larrecq, J., Palamidessi, C., Troina, A.: A probabilistic applied pi-calculus. In: APLAS. (2007) 175–190

13. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations. In: CCS. (2003) 220–230
14. Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW. (2004) 204–218
15. Backes, M., Pfitzmann, B., Waidner, M.: Symmetric authentication in a simulatable Dolev-Yao style cryptographic library. Int. J. Inf. Sec. **4**(3) (2005) 135–154
16. Backes, M., Pfitzmann, B.: A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In: FSTTCS. (2003) 1–12
17. Backes, M., Dürmuth, M.: A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In: CSFW. (2005) 78–93
18. Backes, M.: A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In: ESORICS. (2004) 89–108
19. Patil, A.: On symbolic analysis of cryptographic protocols. Master's thesis, Massachusetts Institute of Technology (2006)
20. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL. (2001) 104–115
21. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theor. Comput. Sci. **367**(1-2) (2006) 2–32

A Translation Rules for Symbolic Interpretation

- Translating internal computations:
  - $symb(\texttt{newnonce()}) = \nu N.\nu x.(\{N/x\}|\_)$
  - $symb(\texttt{encrypt}(m, \texttt{A})) = \nu x.(\{\texttt{enc}(m, PK_A)/x\}|\_)$
  - $symb(\texttt{decrypt}(c)) = \nu x.(\{\texttt{dec}(m, SK_A)/x\}|\_)$
  - $symb(\texttt{sign}(m)) = \nu x.(\{\texttt{sign}(m, SigK_A)/x\}|\_)$
  - $symb(\texttt{verify}(m, s, \texttt{A})) = \nu x.(\{\texttt{ver}(m, s, VerK_A)/x\}|\_)$
  - $symb(\texttt{pair}(m_1, m_2)) = \nu x.(\{\texttt{pair}(m_1, m_2)/x\}|\_)$
  - $symb(\texttt{fst}(m)) = \nu x.(\{\texttt{fst}(m)/x\}|\_)$
  - $symb(\texttt{snd}(m)) = \nu x.(\{\texttt{snd}(m)/x\}|\_)$
- Translating external interactions:
  - $symb(\texttt{input}(x)) = input_{\texttt{R}}(x).(\_)$
  - $symb(\texttt{output}(m)) = \overline{output_{\texttt{R}}}\langle m\rangle.(\_)$
  - $symb(\texttt{send}(m)) = \overline{send_{\texttt{R}}}\langle m\rangle.(\_)$
  - $symb(\texttt{recv}(x)) = recv_{\texttt{R}}(x).(\_)$
  - $symb(\texttt{send}^{\texttt{a}}(m, \texttt{A})) = \overline{send_{\texttt{RA}}}\langle m\rangle.(\_)$
  - $symb(\texttt{recv}^{\texttt{a}}(x, \texttt{A})) = recv_{\texttt{AR}}(x).(\_)$
  - $symb(\texttt{coin}^{\texttt{i}}(b, \texttt{A})) = \overline{toss^i_{\texttt{RA}}}\langle\rangle.toss^i_{\texttt{RA}}(b).(\_)$
  - $symb(\texttt{coin}^{\texttt{r}}(b, \texttt{A})) = \overline{toss^r_{\texttt{RA}}}\langle\rangle.toss^r_{\texttt{RA}}(b).(\_)$
- Translating control flow:
  - $symb(\texttt{P}_1; \texttt{P}_2) = symb(\texttt{P}_1)[symb(\texttt{P}_2)]$
  - $symb(\texttt{if } M = N \texttt{ then } \texttt{P}_1 \texttt{ else } \texttt{P}_2)$
    $= \texttt{if } M = N \texttt{ then } symb(\texttt{P}_1) \texttt{ else } symb(\texttt{P}_2)$
  - $symb(\texttt{prob}(\texttt{P}_1; \texttt{P}_2)) = symb(\texttt{P}_1) \oplus_{0.5} symb(\texttt{P}_2)$
- Key distribution:
  - $PK_R[\_] ::= \nu SK_R.\,(\{\mathsf{pk}(SK_R)/PK_R\} \mid \_)$
  - $SigK_R[\_] ::= \nu SigK_R.\,(\{\mathsf{vk}(SigK_R)/VerK_R\} \mid \_)$
- Finally, $S_{\texttt{R}} ::= SigK_R\,[PK_R\,[symb(\mathcal{P}(\texttt{R}))[\mathsf{0}]]]$

Examples For protocol in Figure 1, the corresponding processes are:

$$
\begin{aligned}
\mathcal{S}_{\mathtt{A}} ::= {} & \nu N.\nu N_a.(\{N/N_a\}| \\
& \nu x_1.(\{\mathtt{pair}(Na, \mathtt{B})/x_1\}| \\
& \nu m_1.(\{\mathtt{enc}(x_1, PK_B)/m_1\}| \\
& \overline{send_{\mathtt{A}}}\langle m_1\rangle. \\
& recv_{\mathtt{R}}(x). \\
& \mathtt{if}\ x = N_a\ \mathtt{then}\ \overline{send_{\mathtt{A}}}\langle\mathtt{true}\rangle))))
\end{aligned}
\qquad
\begin{aligned}
\mathcal{S}_{\mathtt{B}} ::= {} & recv_{\mathtt{B}}(m_1'). \\
& \nu x_1'.(\{\mathtt{dec}(m, SK_B)/x_1'\}| \\
& \nu N.(\{\mathtt{fst}(m_1')/x_1'\}| \\
& \overline{send_{\mathtt{A}}}\langle N\rangle))
\end{aligned}
$$

B Definition of $\mathtt{tr}(\cdot)$ in Section 6

- $\mathtt{tr}\left(\overline{send_{\mathtt{A}}}\langle M\rangle\right) = [\text{“}\textit{send}\text{”}, \mathtt{A}, M]$
- $\mathtt{tr}\left(recv_{\mathtt{A}}(M)\right) = [\text{“}\textit{adv-deliver}\text{”}, \mathtt{A}, M]$
- $\mathtt{tr}\left(\overline{adv^{a}_{\mathtt{AB}}}\langle M, c\rangle\right) = [\text{“}\textit{auth-send}\text{”}, \mathtt{A}, \mathtt{B}, M]$
- $\mathtt{tr}\left(input_{\mathtt{A}}(M)\right) = [\text{“}\textit{input}\text{”}, \mathtt{A}, M]$; $\mathtt{tr}\left(\overline{output_{\mathtt{A}}}\langle M\rangle\right) = [\text{“}\textit{output}\text{”}, \mathtt{A}, M]$
- $\mathtt{tr}\left(\overline{adv^{i}_{\mathtt{AB}}}\langle c, b\rangle\right) = [\text{“}\textit{toss-i}\text{”}, \mathtt{A}, \mathtt{B}, b]$; $\mathtt{tr}\left(\overline{adv^{r}_{\mathtt{AB}}}\langle c, b\rangle\right) = [\text{“}\textit{toss-r}\text{”}, \mathtt{A}, \mathtt{B}, b]$
- $\mathtt{tr}\left(c(M)\right) = [\text{“}\textit{adv-auth}\text{”}, \mathtt{A}, \mathtt{B}, M]$, if c occurs in $\overline{adv^{a}_{\mathtt{AB}}}\langle M, c\rangle$ previously.
- $\mathtt{tr}\left(c()\right) = [\text{“}\textit{adv-tossed-i}\text{”}, \mathtt{A}, \mathtt{B}, b]$, if c occurs in $\overline{adv^{i}_{\mathtt{AB}}}\langle c, b\rangle$ previously.
- $\mathtt{tr}\left(c()\right) = [\text{“}\textit{adv-tossed-r}\text{”}, \mathtt{A}, \mathtt{B}, b]$, if c occurs in $\overline{adv^{r}_{\mathtt{AB}}}\langle c, b\rangle$ previously.

C Rules for Defining Hybrid Traces

- At the beginning, the trace $\mathtt{t}$ is empty.
- If the environment provides m as input to party $\mathtt{R}$, then append trace $\mathtt{t}$ with event $\mathtt{E} = \langle\text{“}\textit{input}\text{”}, \mathtt{R}, m\rangle$.
- If the adversary is activated:
  - If it delivers a message m to party $\mathtt{R}$, then let $\mathtt{E} = \langle\text{“}\textit{adv-deliver}\text{”}, \mathtt{R}, m\rangle$.
  - If it delivers a authenticated message m from party $\mathtt{S}$ to $\mathtt{R}$, then let $\mathtt{E} = \langle\text{“}\textit{adv-auth}\text{”}, \mathtt{S}, \mathtt{R}, m\rangle$.
  - If it delivers coin-tossing result b to initiator $\mathtt{S}$ with responder $\mathtt{R}$, then let $\mathtt{E} = \langle\text{“}\textit{adv-tossed-i}\text{”}, \mathtt{S}, \mathtt{R}, b\rangle$
  - If it delivers coin-tossing result b to responder $\mathtt{R}$ with initiator $\mathtt{S}$, then let $\mathtt{E} = \langle\text{“}\textit{adv-tossed-r}\text{”}, \mathtt{S}, \mathtt{R}, b\rangle$
- If party $\mathtt{R}$ is activated:
  - If it outputs to the environment message m, then let $\mathtt{E} = \langle\text{“}\textit{output}\text{”}, \mathtt{R}, m\rangle$.
  - If its send message m over insecure network, then let $\mathtt{E} = \langle\text{“}\textit{send}\text{”}, \mathtt{R}, m\rangle$.
- If $\mathcal{F}_{\mathrm{AUTH}}$ is activated, and $\mathtt{S}$ wants to send m to $\mathtt{R}$, then let $\mathtt{E} = \langle\text{“}\textit{send-auth}\text{”}, \mathtt{S}, \mathtt{R}, m\rangle$.
- If $\mathcal{F}_{\mathrm{CT}}$ is activated,
  - $\mathtt{S}$ wants to initiate coin-tossing with $\mathtt{R}$, then let $\mathtt{E} = \langle\text{“}\textit{toss-i}\text{”}, \mathtt{S}, \mathtt{R}\rangle$.
  - $\mathtt{R}$ acts as the responder with $\mathtt{S}$, then let $\mathtt{E} = \langle\text{“}\textit{toss-r}\text{”}, \mathtt{S}, \mathtt{R}\rangle$.
- If $\mathcal{F}_{\mathrm{CPKE}}$ is activated,
  - If $\mathtt{S}$ encrypts with $(\mathtt{Encrypt}, \mathtt{PID}_{\mathtt{R}}, m)$, and $\mathcal{F}_{\mathrm{CPKE}}$ returns c, then let $\mathtt{E} = \langle\text{“}\textit{ciphertext}\text{”}, \mathtt{PID}_{\mathtt{R}}, m, c\rangle$.
  - If $\mathtt{S}$ decrypts with $(\mathtt{Decrypt}, \mathtt{PID}_{\mathtt{S}}, c)$, and $\mathcal{F}_{\mathrm{CPKE}}$ returns m, then let $\mathtt{E} = \langle\text{“}\textit{decrypt}\text{”}, \mathtt{PID}_{\mathtt{S}}, c, m\rangle$.
- The case for $\mathcal{F}_{\mathrm{CERT}}$ is similar.