Symmetric encryption in automatic analyses for confidentiality against active adversaries

Peeter Laud
Tartu University and Cybernetica AS
Email: peeter_l@ut.ee

Abstract— **In this article we present a technique for static analysis, correct with respect to complexity-theoretic definitions of security, of cryptographic protocols for checking whether these protocols satisfy confidentiality properties. The approach is similar to Abadi and Rogaway — we define *patterns* for cryptographic protocols (they did it for formal expressions), such that the protocol is secure iff the patterns are. We then statically analyse the patterns, they should be easier to analyse than the protocols themselves. We consider symmetric encryption as the cryptographic primitive in protocols. Handling this primitive has so far received comparatively less attention in approaches striving to unite the formal and computational models of cryptography.**

I. Introduction

A cryptographic protocol is expected to satisfy certain security properties. In this paper we are interested in confidentiality properties which state that all protocol runs must look somehow similar — the view of the adversary may not depend on the secret information that one legitimate protocol party wishes to transmit to some other one.

Protocols have to be verified about satisfying the required security properties. The verification procedure should produce some kind of proof that the property is satisfied. It is possible to craft these proofs by hand. However, there exist a lot of protocols, so automatic verification methods are necessary. Such methods do indeed exist; there are tools which take protocol descriptions as inputs and return whether these protocols are secure or not (and in the case of insecurity, may even show an attack). The methods do not necessarily always give a definite answer (the problem of deciding whether the protocol is secure may be very hard, depending on the definition of security), they may also have an option to answer "don't know". Even methods that sometimes answer "don't know" are useful if they can be expected to give definite answers for reasonable protocols reasonably often. Quite often the methods have only two possible answers — "secure" and "don't know". This is called "erring on the safe side" — labeling an insecure protocol secure may have much more serious consequences than labeling a secure protocol potentially insecure.

Almost all the existing tools and methods for automatic checking of security properties have one common shortcoming — they are based on the Dolev-Yao model, i.e. they assume "perfect" cryptography. In this model, the exchanged messages are modeled as elements of a certain term algebra; the capabilities of the adversary and the security definitions are also given in terms of that algebra.

In real life, and in the *computational* model, messages are not formal terms, but are bit-strings. The adversary can be any efficient algorithm. In theory, there is still a gap between the Dolev-Yao model and the computational model. It is not known how much of the results obtained for the Dolev-Yao model carry over to the computational model. Abadi and Rogaway [1] have started bridging that gap by defining *patterns* for formal messages (built up from atomic messages, pairing and symmetric encryption), such that the computational interpretations of the message and its pattern are indistinguishable. The pattern of a formal message has simpler structure than the message itself. One could use that pattern to determine whether something about some secret atomic message M can be found from a given message — if M does not occur in the pattern of that message then the interpretation of that message surely does not leak M. Abadi and Jürjens [2] have later extended this line of work, basically giving a protocol analysis whose results are correct in the computational model if we only consider passive adversaries.

Our approach is similar. Given a protocol, we construct a number of protocols with simpler structure; this set of constructed protocols takes the role of the *pattern*. We show that if none of the constructed protocols leak M, then the original protocol does not leak M. A protocol surely does not leak M if it has not been used in the construction of any messages exchanged during the run of this protocol; we use that criterion to decide the security of constructed protocols. Typically, if the secret M occurred only under encryption in the messages exchanged in the original protocol (and this protocol was secure), then it does not occur at all in the constructed protocols; hence this criterion is sufficiently powerful for our purposes. An earlier version of the analysis has been implemented [3].

This paper has the following structure. After reviewing some related work in Sec. II and basic definitions in Sec. III we state what properties the encryption system has to satisfy in Sec. IV and what a protocol is in Sec. V. We give the security definition of the protocol in Sec. VI. In Sec. VII we describe our main contribution — a protocol transformation that makes the protocols more easily analyzable. The correctness of this transformation is based on the security definition of symmetric encryption. This contribution gives a completely automatable method of analyzing protocols for the preservation of confidentiality of secret messages. The results of the analysis are correct in the computational model, even when considering *active* adversaries. Last, we describe a very simple and conservative information flow analysis in Sec. VIII that is nevertheless quite successful in analyzing the transformed protocols. We give an example of the transformation (and subsequent information flow analysis) in Sec. IX.

The full version of this paper is available as [4].

II. Related work

The automated cryptographically sound analysis of the usage of cryptographic primitives is a rather new research topic. Some work has been done in the area of considering only passive adversaries by ourselves [5], [6] and before that, by Abadi et al. [1], [2]. These papers have presented analyses for secure information flow (or something equivalent to that) for programs written in a certain language containing cryptographic primitives. Particularly, [2] gave the means to automatically verify when a protocol preserves confidentiality against passive adversaries. As we already explained before, our work may be seen as an extension of [1], [2]. In some sense, we are probing here, how well their approach extends to the setting with active attacks. Similarly to them, we cannot handle unbounded replication yet (although the language in [2] contains the replication operator, we have explained in [5, Chapter 7] why that does not count).

Backes, Pfitzmann et al. [7]–[11] have given a framework for faithful abstraction of cryptographic primitives, in a very general setting, allowing to model all the details how a cryptographic primitive may or may not operate. This framework allows them to reason about protocols, considering only the abstract primitives. The "modeling of all details" is achieved by introducing an *honest user* into the system, interacting with both the primitive and the adversary. The abstraction of the primitive is a good one if there exists a transformation of adversaries, such that no honest user can distinguish the interaction with the real primitive and any adversary from the interaction with the abstract primitive and the transformed version of that adversary; this notion is called *simulatability*. They have not presented any fully automatic analyses, though. Recently, they have defined a cryptographic library [12] with abstract cryptographic operations and their implementations, and have proved that if a protocol is secure with respect to abstract semantics then it is also secure with respect to concrete semantics. The abstract semantics is rather similar to the Dolev-Yao model (although not exactly the same). In some sense, the adversary attacking the library with respect to the abstract semantics is even less powerful than the Dolev-Yao adversary, because it may not be nondeterministic. In our opinion, it would be quite easy to add a Dolev-Yao analyzer, for example Blanchet's [13], to the framework and show that its results are cryptographically correct.

Our results are still not subsumed by a combination of [12] and [13]. We consider a different cryptographic primitive — symmetric encryption, while the library [12] contains public-key encryption and signing. Our main improvement is in the handling of secret keys. We put no restrictions on it. The library [12] only allows secret keys to be used for decryption or signing, one may not put them to other expressions (neither does a recent addendum [14] lift any of these restrictions put on secret *decryption* keys). This constraint is nontrivial, removing it means that one has to deal with *encryption cycles* [1] and also with the *commitment problem* described below. An encryption cycle occurs when the adversary knows the encryption of one (secret) key under the second, the encryption of the second secret key under the third, etc., and the encryption of the last secret key under the first. The computational security definitions do not say anything about them — the setup of these definitions does not allow the adversary to construct encryption cycles. We deal with encryption cycles by replacing the concrete encryption primitive with the abstract primitive one key at a time.

The determinism of the adversary attacking the library with respect to the abstract semantics is obviously not a weakness of the results of Backes et al., but it may hamper the further extension of these results. If it turns out that the Dolev-Yao adversary really *must* be non-deterministic to achieve its full power, then non-determinism also has to be built into the definitions of simulatability. When conceiving simulatable abstractions to symmetric encryption, one also has to account for the *commitment problem* that occurs when the simulator first receives a ciphertext and only later the corresponding plaintext and the key. In [15], Backes and Pfitzmann actually give a simulatable abstraction of symmetric encryption, but under the condition that neither the commitment problem nor the encryption cycles occur.

Canetti [16] has given a framework similar to Backes, Pfitzmann et al. (some details differ) with the same goals. He also postulates that an analysis similar to the one presented here may be possible. Several cryptographic primitives have been presented in this framework, for example commitments [17], secure channels [18] and multiparty computation [19], although nothing similar to a cryptographic library. Apparently the focus has been more on constructing protocols than analyzing them.

Herzog [20] has given a computational implementation of the Dolev-Yao model (with public-key encryption as the only cryptographic primitive) that actually respects all the properties of the formal model, i.e. if a protocol is secure in the formal model then its implementation is secure in the computational model. The downside of this result is the extremely strong security condition on the encryption — it must be *plaintext aware* [21], this property is achievable only in the random oracle model [22] (Recently, Herzog et al. [23] have given a definition of plaintext awareness that is sufficient for their purposes and does not require a random oracle, but needs a third party). In our work, the security requirements put on the encryption primitive (security against adaptive CCA and ciphertext integrity) are achievable under the standard assumptions [24]. Herzog also does not handle encryption cycles, they are simply assumed not to occur.

III. Basic Cryptographic Notions

A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if for all positive polynomials p there exists $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $|f(n)| < 1/p(n)$.

The cryptographic algorithms work with bit-strings, we denote the set of all bit-strings with Σ. An algorithm $\mathcal{A}$ takes a number of arguments and returns a result. It runs in *probabilistic polynomial-time (PPT)* if there exists a polynomial p such that the number of steps made by $\mathcal{A}$ is bounded by $p(n)$ where n is the length of the first argument of $\mathcal{A}$. In the following, the first argument will always be the *security parameter*, indicating the desired security of the cryptographic algorithms (more security also means longer running times). As the definition of probabilistic polynomial-time spoke about the *length* of the first argument, the security parameter is given to the algorithms in *unary*, denoted by 1^n.

A probability distribution over some finite or countable set X is a function $\mu : X \to [0, 1]$, such that $\sum_{x\in X} \mu(x) = 1$. We denote the set of all probability distributions over the set X by $\mathcal{D}(X)$. We denote picking the value of the random variable x according to the distribution $D \in \mathcal{D}(X)$ by $x \leftarrow D$. The notation $\{\!|E : C|\!\}$ denotes the distribution of the random variable or expression E with the probability space defined by C, here C is a list of picking values to random variables and of defining variables.

Two *families* of probability distributions $D = \{D_n\}_{n\in\mathbb{N}}$ and $D' = \{D'_n\}_{n\in\mathbb{N}}$, where $D_n, D'_n \in \mathcal{D}(\Sigma)$, are *indistinguishable*, denoted $D \approx D'$, if for all PPT algorithms $\mathcal{A}$, the difference of probabilities

$$\mathbf{P}[b = 1 \,|\, x \leftarrow D_n, b \leftarrow \mathcal{A}(1^n, x)] - \mathbf{P}[b = 1 \,|\, x \leftarrow D'_n, b \leftarrow \mathcal{A}(1^n, x)]$$

is a negligible function of n. Here the supposed role of $\mathcal{A}$ is to understand whether its second argument was distributed according to D_n (then $\mathcal{A}$ should return 1) or according to D'_n (then $\mathcal{A}$ should return something else). Indistinguishability is the cryptographic equivalent of sameness. It is an equivalence relation over the set of families of probability distributions over Σ.

An algorithm $\mathcal{A}$ does not have to be fully specified. Instead, it may make queries to an *oracle*. An oracle $\mathcal{O}$ is again a probabilistic algorithm. The algorithm $\mathcal{A}$ may use special commands to query $\mathcal{O}$ with certain arguments. When queried, $\mathcal{O}$ runs and returns the result back to $\mathcal{A}$. We denote $\mathcal{A}$, making queries to $\mathcal{O}$, with $\mathcal{A}^{\mathcal{O}}$. When we want to stress that $\mathcal{A}$ may make queries to an oracle then we write it as $\mathcal{A}^{(\cdot)}$. When we want to stress that $\mathcal{O}$ takes an argument, we write $\mathcal{A}$ together with $\mathcal{O}$ as $\mathcal{A}^{\mathcal{O}(\cdot)}$. The notation $\mathcal{A}^{(\cdot),(\cdot)}$ means that $\mathcal{A}$ may make queries to two different oracles.

The notion of algorithms with oracles allows us to generalize the notion of indistinguishability to algorithms. We say that two algorithms (or oracles; or black boxes) $\mathcal{O}$ and $\mathcal{O}'$ are indistinguishable if the difference of probabilities

$$\mathbf{P}[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathcal{O}(1^n,\cdot)}(1^n)] - \mathbf{P}[b = 1 \,|\, b \leftarrow \mathcal{A}^{\mathcal{O}'(1^n,\cdot)}(1^n)]$$

is negligible in n for all PPT algorithms $\mathcal{A}$. Instead of algorithms we could have also considered families of probability distributions over algorithms.

IV. Security of encryption systems

An *encryption system* is a triple of algorithms $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. Here $\mathcal{K}$ is the *key generation algorithm*, $\mathcal{E}$ is the *encryption algorithm* and $\mathcal{D}$ is the *decryption algorithm*. The algorithms $\mathcal{K}$ and $\mathcal{E}$ are probabilistic, the algorithm $\mathcal{D}$ is deterministic. All algorithms take the *security parameter* n (represented in unary — 1^n) as an argument. Additionally, $\mathcal{E}$ takes two more arguments — the key and the plaintext. Also, $\mathcal{D}$ takes two more arguments — the key and the ciphertext. Let $x \in \Sigma$. For all k that may be returned by $\mathcal{K}(1^n)$ and for all y that may be returned by $\mathcal{E}(1^n, k, x)$, the algorithm $\mathcal{D}(1^n, k, y)$ must return x.

The previous requirement stated only that $\mathcal{D}$ is the inverse of $\mathcal{E}$. We also want the encryption system to be secure, i.e. we want the ciphertext to not give away any information about the plaintext. More concretely, we want the encryption system to satisfy the following properties. If an encryption system satisfies these properties, then we call it *secure*.

First, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ must be secure against adaptive chosen-ciphertext attacks (CCA). This means that no (probabilistic polynomial-time) adversary $\mathcal{A}^{(\cdot),(\cdot)}$ that does not query its second oracle with the values returned by the first is unable to distinguish the pair of black boxes $\mathcal{E}(1^n, k, \cdot)$ and $\mathcal{D}(1^n, k, \cdot)$ from the pair of black boxes $\mathcal{E}(1^n, k, 0^{|\cdot|})$ and $\mathcal{D}(1^n, k, \cdot)$. Here k is distributed according to $\mathcal{K}(1^n)$ and $0^{|\cdot|}$ denotes the bit-string of all zeroes with the same length as the argument to the black box. Note that with such security definition, the length of the ciphertext must depend only on the length of the plaintext (and the security parameter).

The preceding definition most closely follows that of Bellare et al. [25] (the variant which they call "real-or-random indistinguishability"), although the notion of security against CCA already appeared much earlier. The notion of non-adaptive CCA first appeared in [26] and adaptive CCA in [27].

Second, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ must have ciphertext integrity. This means that for all probabilistic polynomial-time adversaries $\mathcal{A}^{(\cdot),(\cdot)}$ there is only a negligible probability that during the run of $\mathcal{A}^{\mathcal{E}(1^n,k,\cdot),\mathcal{D}(1^n,k,\cdot)}$, the oracle $\mathcal{D}$ is queried with a bit-string x that is a valid ciphertext and was not previously returned by the oracle $\mathcal{E}$. Again, k is distributed according to $\mathcal{K}(1^n)$. This definition comes from [24].

Third, we want the algorithm $\mathcal{E}$ to return a different bit-string each time it is queried. In the absence of encryption cycles, this property easily follows from the security against chosen-ciphertext attacks, but we want this property to hold universally.

The requirement for the encryption to always return different results can be formalized as follows. Consider the oracle $\mathcal{E}(1^n, \cdot, \cdot)$. This oracle simply invokes the encryption algorithm but it takes *both* the plaintext *and the key* as arguments (in contrast to the oracle $\mathcal{E}(1^n, k, \cdot)$). We require that no adversary can cause (with non-negligible probability) the oracle $\mathcal{E}(1^n, \cdot, \cdot)$ to give the same answer to two queries.

An encryption system satisfying the last requirement can be constructed as follows. Let $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ be any encryption system secure against CCA and having ciphertext integrity (such systems exist under standard assumptions, see [24]). We construct an encryption system $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ as follows.

- $\mathcal{K}$ is equal to $\mathcal{K}'$.
- $\mathcal{E}(1^n, k, x)$ first generates a random bit-string r of length n. It then constructs $y \leftarrow \mathcal{E}'(1^n, k, (x, r))$ and outputs (y, r).
- $\mathcal{D}(1^n, k, (y, r))$ first computes $z = \mathcal{D}'(1^n, k, y)$. If z is not a pair of bit-strings (x, r'), then it outputs $\perp$. Otherwise it checks whether $r = r'$. If yes, then it outputs x, otherwise it outputs $\perp$.

Obviously the probability that $\mathcal{E}$ returns the same answer twice is bounded from above by $q^2/2^n$, where q is the number of queries that the adversary makes to $\mathcal{E}(1^n, \cdot, \cdot)$. It is negligible, because q is at most polynomial in n.

Theorem 1: $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is secure against CCA and has ciphertext integrity.

Proof: First we show that $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is secure against CCA. Assume the opposite, then there exists an adversary $\mathcal{A}$ that can distinguish

$$(\mathcal{E}(1^n, k, \cdot), \mathcal{D}(1^n, k, \cdot)) \text{ from } (\mathcal{E}(1^n, k, 0^{|\cdot|}), \mathcal{D}(1^n, k, \cdot))$$

without passing the output of the first black box to the second. We can now very easily construct an adversary $\mathcal{A}'$ that can distinguish

$$((\mathcal{E}'(1^n, k, (\cdot, r)), r), \mathcal{D}'(1^n, k, \cdot)) \text{ from } ((\mathcal{E}'(1^n, k, (0^{|\cdot|}, r)), r), \mathcal{D}'(1^n, k, \cdot)) \quad (1)$$

without passing the first part of the output of the first black box to the second. The adversary $\mathcal{A}'$ works just like $\mathcal{A}$, but additionally it implements the definition of $\mathcal{D}$ in terms of $\mathcal{D}'$. On the other hand, no adversary can distinguish

$$((\mathcal{E}'(1^n, k, (0^{|\cdot|}, r)), r), \mathcal{D}'(1^n, k, \cdot)) \text{ from } ((\mathcal{E}'(1^n, k, 0^{|(\cdot, r)|}), r), \mathcal{D}'(1^n, k, \cdot)) \quad (2)$$

without passing the first part of the output of the first black box to the second, this follows directly from the security of $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ against CCA. Indeed, (2) corresponds just to a certain choice of bit-strings submitted to the first black box. These bit-strings are pairs of some bit-string consisting of zeroes only and a random bit-string. As $\mathcal{A}'$ can distinguish (1) but nobody can distinguish (2), $\mathcal{A}'$ can also distinguish

$$((\mathcal{E}'(1^n, k, (\cdot, r)), r), \mathcal{D}'(1^n, k, \cdot)) \text{ from } ((\mathcal{E}'(1^n, k, 0^{|(\cdot, r)|}), r), \mathcal{D}'(1^n, k, \cdot)) \quad (3)$$

without passing the first part of the output of the first black box to the second. But this violates the security of $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ against CCA. Indeed, (3) corresponds just to a certain choice of bit-strings submitted to the first black box. These bit-strings are pairs where the second component is a random bit- string.

Let us show now that $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ has ciphertext integrity. Assume the opposite, then there exists an adversary $\mathcal{A}$ that in interaction with black boxes $\mathcal{E}(1^n, k, \cdot)$ and $\mathcal{D}(1^n, k, \cdot)$ can construct a valid ciphertext that it has not seen before. Then there exists an adversary $\mathcal{A}'$ (it works like $\mathcal{A}$, but additionally implements the definitions of $\mathcal{E}$ and $\mathcal{D}$) that in interaction with black boxes $\mathcal{E}'(1^n, k, \cdot)$ and $\mathcal{D}'(1^n, k, \cdot)$ can construct a pair (y, r), where y is a valid ciphertext that decrypts to (x, r) under $\mathcal{D}'(1^n, k, \cdot)$. Also, $\mathcal{A}'$ has not seen the pair (y, r) before. Now, if y had been an output of $\mathcal{E}'(1^n, k, \cdot)$ before, then the adversary $\mathcal{A}$ would have obtained a pair (y, r') at this point as an output of $\mathcal{E}(1^n, k, \cdot)$. Also, here $r \neq r'$, as (y, r) was the new valid ciphertext constructed by $\mathcal{A}$. But it is impossible to obtain (y, r') from $\mathcal{E}(1^n, k, \cdot)$, as it is not a valid ciphertext of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ under the key k. Indeed, $\mathcal{D}'(1^n, k, y) = (x, r)$ and $r \neq r'$. ■

V. Syntax and semantics of protocols

A protocol consists of a set of participants. Each participant is basically a program in a certain programming language. The programs consist of computing certain values from the ones already known to participants, exchanging messages and checking the values of received messages. Additionally, there are some long-term secrets that the participants know and/or share; these long-term secrets make the identification of different participants possible and they have been exchanged before the start of the protocol (we model their exchange by an extra "prelude" to the protocol).

For concreteness, we let a program be a sequence of statements where statements are defined in Fig. 1. The meaning of most statements should be clear (we give a precise semantics of the protocol below). Let us just mention that the superscript ℓ at the $\mathbf{receive}^\ell$-statement denotes the expected length of the received message. Both ℓ and b at $\mathbf{constant}(b)$ are functions of the security parameter. The recipient of $\mathbf{send}\ x$ is not specified because all communication is under the control of the adversary anyway. The message constructors (key generation, tupling, encryption, constant and random number generation) tag the constructed message with the name of that constructor (the adversary is obviously free to generate all possible messages). If something goes wrong at the execution of a statement then the participant containing that statement becomes stuck.

$$\begin{array}{rcll}
P & ::= & k := \mathbf{gen_key} & \mid\ y := (x_1, \ldots, x_m) \\
 & \mid & x := \pi_i^m(y) & \mid\ x := encr_k(y) \\
 & \mid & y := decr_k(x) & \mid\ x := \mathbf{random} \\
 & \mid & \mathbf{send}\ x & \mid\ x := \mathbf{receive}^\ell \\
 & \mid & \mathbf{check}(x = y) & \mid\ x := \mathbf{constant}(b) \\
 & \mid & x := y &
\end{array}$$

$k, x, x_1, \ldots, x_m, y$ are *variables* from a set $\mathbf{Var}$.

Fig. 1. Statements

We assume that in a protocol, each variable occurs at most once at the left side of an assignment. Let $\mathbf{Var}_{\mathrm{sent}}$ be the set of all variables x, such that the statement **send** x occurs in the protocol.

Currently a shortcoming of our work is its inability to handle replication, i.e. we can only analyse bounded number of protocol instances running simultaneously. The difficulties in analyzing arbitrary number of instances lay in the area of static analysis, not in the area of cryptography (in constructing new protocols, we have to keep track of all produced values, if we have no loops then this is simpler because each produced value has a unique name). The program language itself is quite minimalistic too, but extending it (for example, with exception handling) should not be overly difficult. The expressiveness of the presented language is rather similar to strand spaces [28].

There are some more statements that do not occur in the original protocols given to our analysis, but may occur in the transformed protocols. Let $\mathbf{Sem}$ be a set of *semaphores*. The statements $signal(s)$ and $wait(s)$ (here $s \in \mathbf{Sem}$) implement a simple synchronization primitive — the statement $signal(s)$ records that the semaphore s has been passed and the statement $wait(s)$ checks whether s has been passed and becomes stuck, if this is not the case. Each semaphore s may occur at most once in the statement $signal(s)$. The intermediate protocols may also contain a *case*-statement of the form

$$\begin{array}{l} w := case\ z\ of \\ \qquad x_1 \to y_1 \\ \qquad \cdots\cdots\cdots \\ \qquad x_n \to y_n \end{array} \tag{4}$$

with the obvious semantics. Here $w, z, x_i, y_i \in \mathbf{Var}$. (Neither the initial nor the final protocols contain the *case*-statement, only the intermediate ones.)

Let $\mathbf{Parts}$ be the set of (the names of) protocol participants. The protocol $\wp$ now is a function mapping each element of $\mathbf{Parts} \dot\cup \{\mathsf{pre}\}$ to a program. If $A \in \mathbf{Parts}$, then $\wp(A)$ is the program executed by the participant A. The protocol also has a "prelude" $\wp(\mathsf{pre})$. It is executed before the rest of the protocol. It is used for initializing the long-term secrets before the "real protocol" starts. Let $\mathbf{Parts}'$ denote the set $\mathbf{Parts} \dot\cup \{\mathsf{pre}\}$. We demand that none of the programs $\wp(A)$, where $A \in \mathbf{Parts}'$, is empty.

The protocol runs in parallel to the adversary. The adversary relays the messages between protocol participants. The adversary also schedules the participants; by letting the adversary do the scheduling we have removed the possible non-determinism (as opposed to probabilism; the semantics of statements is obviously probabilistic) arising from the parallel execution of participants. Overall, the computation is rather similar to [29].

Let $\mathbf{ASt}$ be the set of all possible internal states of the adversary; the set $\mathbf{ASt}$ is (isomorphic to) some subset of Σ. If the security parameter has been set to n, then the adversary $\mathcal{A}(1^n, \ldots)$ implements a probabilistic function $\mathcal{A}_n$ from $\Sigma_\perp \times \mathbf{ASt}$ to $\Sigma \times \mathbf{Parts}' \times \mathbf{ASt}$; the running time of $\mathcal{A}(1^n)$ must be polynomial in n. Here $X_\perp$ denotes the set $X \dot\cup \{\perp\}$. Here the first argument to $\mathcal{A}_n$ is the message that has been sent by some participant at the latest moment (if any). The first return value of $\mathcal{A}_n$ is the message received by a protocol party executing $x :=$ **receive** at the next moment; the second return value is the name of the protocol participant that executes next.

We will now give a precise description of the execution of the protocol $\wp$. During its run, the values will be assigned to some or all of the variables. Let $\mathsf{M} \in \Sigma$ be the secret message and $M \in \mathbf{Var}$ a distinguished variable. We assume that $|\mathsf{M}|$ is public knowledge, because our encryption system does not hide the lengths of plaintexts. Let $D_n^M \in \mathcal{D}(\Sigma)$ denote the distribution of M for the security parameter n.

Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a fixed secure encryption system such that all possible keys produced by $\mathcal{K}(1^n)$ (for a fixed n) have the same length. Let p be a polynomial and let τ be an efficiently computable injective function from $\Sigma \times \Sigma$ to Σ, such that τ^{-1} is also efficiently computable and the length of $\tau(b_1, b_2)$ depends only on the lengths of b_1 and b_2. Let $\mathsf{ciphertext}$, key, tuple, random, secret and const be fixed bit-strings. The *running* of the protocol $\wp$ together with the adversary $\mathcal{A}$ proceeds as follows.

Initialization. Some auxiliary variables are necessary for the description of running the protocol.

- The variable sm contains the message that is about to be sent out. We initialize this to $\perp$ (i.e. in the beginning, there is no message waiting).
- The variable rm contains the message that has been received from the network (i.e. the adversary). We initialize this to ε (the empty bit-string).
- The programs $\mathsf{P}(A)$, where $A \in \mathbf{Parts}'$ contain the yet unexecuted parts of the programs. We initialize $\mathsf{P}(A)$ with $\wp(A)$.
- The flags $r_A \in \{\mathsf{true}, \mathsf{false}\}$, where $A \in \mathbf{Parts}'$ indicate whether the participant A is still running or it has become stuck. We initialize them with true.
- AS is the internal state of the adversary. We initialize it with ε.
- C is the participant that is about to execute the next statement. We initialize it with pre.
- For all variables $x \in \mathbf{Var}$ occurring in $\wp$, the flag i_x shows whether this variable has been already initialised. We initialize i_M with true and the rest with false.
- For all variables $x \in \mathbf{Var}$ occurring in $\wp$, their value is recorded in v_x. We initialize v_M with $\tau(\mathsf{secret}, \mathsf{M})$ and leave the rest of v_x uninitialized.
- For all semaphores s occurring in $\wp$, the flag p_s shows whether this semaphore has been passed. We initialize these flags to false.

We call the sequence of all these values an *execution state*. We denote the components of an execution state E by $E[sm]$, $E[rm]$, $E[AS]$, etc. Let $\mathbf{ExcSt}$ be the set of all execution states.

Step. A step describes the execution of one statement of one program of the protocol. The running of the protocol consists of executing the steps until an instruction to stop the execution has been reached. During a step, the following sub-steps are done:

1) Consider C. If one of the following conditions is true then stop the execution of the protocol.

- $\mathsf{P}(C)$ is an empty sequence of statements.
- $C \neq \mathsf{pre}$ and $\mathsf{P}(\mathsf{pre})$ is not an empty sequence of statements.

2) Let S be the first statement of $\mathsf{P}(C)$ and set $\mathsf{P}(C)$ to the rest of $\mathsf{P}(C)$.
3) If r_C is *false* (i.e. the participant C is stuck), then set sm to $\perp$ and continue with sub-step 5.
4) Consider the statement S. If it is not **send** then set sm to $\perp$. If it is not a *case*-statement and i_x is *false* for any variable whose value is used by that statement then set r_C to *false* and continue with sub-step 5. Otherwise,
   - If S is $k := \mathbf{gen\_key}$, then invoke $r := \mathcal{K}(1^n)$ and set v_k to $\tau(\mathsf{key}, r)$. Also, set i_k to true.
   - If S is $y := (x_1, \ldots, x_m)$ then set v_y to
     $$\tau(\tau(\mathsf{tuple}, m), \tau(v_{x_1}, \tau(\cdots \tau(v_{x_{m-1}}, v_{x_m})) \cdots)$$
     and i_y to true.
   - If S is $x := \pi_i^m(y)$, then parse v_y as:
     $$\begin{array}{rcl} (t, z_0) & := & \tau^{-1}(y) \\ (x_1, z_1) & := & \tau^{-1}(z_0) \\ (x_2, z_2) & := & \tau^{-1}(z_1) \\ \multicolumn{3}{c}{\cdots\cdots\cdots\cdots\cdots\cdots} \\ (x_{m-1}, z_{m-1}) & := & \tau^{-1}(z_{m-2}) \end{array}$$
     and set $x_m = z_{m-1}$. If one of the following conditions holds then set r_C to false:
     - Some invocation of τ^{-1} fails.
     - $t \neq \tau(\mathsf{tuple}, m)$.
     - x_i is not *properly tagged* — when τ^{-1} is applied to it then the first component of the result is not one of ciphertext, key, random, secret, const or $\tau(\mathsf{tuple}, m')$ for some $m' \in \mathbb{N}$.

     Otherwise set v_x to x_i and i_x to true.
   - If S is $x := encr_k(y)$ then parse v_k as $(t, k) := \tau^{-1}(v_k)$. If the invocation of τ^{-1} fails or t is not key then set r_C to false. Otherwise invoke $r := \mathcal{E}(1^n, k, v_y)$ and set v_x to $\tau(\mathsf{ciphertext}, r)$. Also, set i_x to true.
   - If S is $y := decr_k(x)$ then parse v_k as $(t_k, k) := \tau^{-1}(v_k)$ and v_x as $(t_x, x) := \tau^{-1}(v_x)$. If some invocation of τ^{-1} fails or t_k is not key or t_x is not ciphertext then set r_C to false. Otherwise invoke $r := \mathcal{D}(1^n, k, x)$. If the invocation fails or r is not properly tagged then set r_C to false. Otherwise set v_y to r and i_y to true.
   - If S is $x := \mathbf{random}$ then let r be a uniformly randomly generated bit-string of length $p(n)$. Set v_x to $\tau(\mathsf{random}, r)$ and i_x to true.
   - If S is $x := \mathbf{constant}(b)$ then set v_x to $\tau(\mathsf{const}, b(n))$ (recall that the type of b was $\mathbb{N} \to \Sigma$) and i_x to true.
   - If S is $\mathbf{send}\ x$ then set sm to v_x.
   - If S is $x := \mathbf{receive}^\ell$ then check whether rm is properly tagged and whether $|rm| = \ell(n)$. If yes, then set v_x to rm and i_x to true, otherwise set r_C to false.
   - If S is $\mathbf{check}(x = y)$ and $v_x \neq v_y$ then set r_C to false.
   - If S is $x := y$ then set i_x to true and v_x to v_y.
   - If S is $signal(s)$ then set p_s to true.
   - If S is $wait(s)$ then check whether p_s is true. If not then set r_C to false.
   - If S is a *case*-statement of the form (4) and i_z is false then set r_C to false. Otherwise find the first index j, such that i_{x_j} is true and $v_z = v_{x_j}$. If there is no such j or if i_{y_j} is false then set r_C to false. Otherwise set v_w to v_{y_j} and i_w to true.
5) If r_{pre} is false then stop the execution of the protocol.
6) Invoke the adversary:
   $$(rm, C, AS) \leftarrow \mathcal{A}_n(sm, AS) \ .$$

The steps define a probabilistic transition system (for each value of the security parameter n) on the set of execution states. Probabilistic effects are introduced first by the invocation of the adversary, which may employ random coin-tosses, and second by certain statements of the protocol (generation of random numbers and new keys and encryption). For all $E, E' \in \mathbf{ExcSt}$ there is a certain probability that the execution transitions from the state E to the state E' in one step. The sum (over E') of all these probabilities is at most 1, if its less than 1, then there is non-zero probability that E is the final state (i.e. execution is stopped in state E). The probabilistic transition system defines certain probability distribution over the set $\mathbf{ExcSt}$, to each E we assign the probability that the execution of $\wp$ together with the adversary $\mathcal{A}$ ends in that state. We call that probability distribution the *semantics* of $\wp$ together with $\mathcal{A}$ and denote it by $[\![\wp, \mathcal{A}]\!]_n(\mathsf{M})$.

Sometimes we want to talk about fixing the random choices made by the protocol. Let $\mathbf{R}$ be the set of all possible random choices made the adversary and the statements of the protocol during the execution. We are now going to describe the structure of the elements of $\mathbf{R}$.

For all $A \in \mathbf{Parts}'$, let l_A be the length of the program $\wp(A)$. Let $L_\wp$ be the following set:

$$L_\wp = \{(A, i) \,|\, A \in \mathbf{Parts}', 1 \leq i \leq l_A\} \ . \tag{5}$$

An element $\mathbf{r}$ of $\mathbf{R}$ is a mapping from $(L_\wp \times \{\mathsf{p}, \mathsf{a}\}) \dot\cup \{M\}$ to Σ. Here $\mathbf{r}(M)$ is the secret value M. The value $\mathbf{r}(A, i, \mathsf{p})$ contains the random coins used by the i-th statement of the participant A and the value $\mathbf{r}(A, i, \mathsf{a})$ contains the random coins used by the adversary right after the execution of the i-th statement of the participant A.

Fixing the random choices gives us a (plain) transition system on the set of execution traces. We denote these transitions (for a fixed $\mathbf{r} \in \mathbf{R}$) by $\xrightarrow{\mathbf{r}}$. Later, this gives us the possibility to talk about two executions of the protocol going in lock-step.

When we have some objects that use random coin-tosses and we want to talk about their value once we have fixed these tosses, then we denote the resulting value by superscripting it with these tosses. For example, $[\![\wp, \mathcal{A}]\!]_n^{\mathbf{r}}$ is the final execution state that we get for the security parameter n and the random coin-tosses $\mathbf{r}$.

Protocol prelude	the program for B
$K_{AS} := \mathbf{gen\_key}^{(1)}$ $K_{BS} := \mathbf{gen\_key}^{(2)}$ $A := \mathbf{constant}(\text{“A”})$ $B := \mathbf{constant}(\text{“B”})$	$m_3 := \mathbf{receive}^{\ell_3\,(10)}$ $dm_3 := \mathit{decr}_{K_{BS}}(m_3)$ $K^B_{AB} := \pi^2_1(m_3)$ $m_4 := \mathbf{receive}^{\ell_4\,(11)}$ $bM := \mathit{decr}_{K^B_{AB}}(m_4)$
the program for A	**the program for S**
$N^A_A := \mathbf{random}^{(3)}$ $\mathbf{send}\ (A, B, N^A_A)$ $m^A_2 := \mathbf{receive}^{\ell_2\,(4)}$ $dm^A_2 := \mathit{decr}_{K_{AS}}(m^A_2)$ $\mathbf{check}(N^A_A = \pi^4_1(dm^A_2))$ $\mathbf{send}\ \pi^4_4(dm^A_2)$ $K^A_{AB} := \pi^4_3(dm^A_2)$ $aM := \mathit{encr}_{K^A_{AB}}(M)^{(5)}$ $\mathbf{send}\ aM$	$m_1 := \mathbf{receive}^{\ell_1\,(6)}$ $N^S_A := \pi^3_3(m_1)$ $K_{AB} := \mathbf{gen\_key}^{(7)}$ $dfb := (K_{AB}, A)$ $fb := \mathit{encr}_{K_{BS}}(dfb)^{(8)}$ $dfa := (N^S_A, B, K_{AB}, fb)$ $fa := \mathit{encr}_{K_{AS}}(dfa)^{(9)}$ $\mathbf{send}\ fa$

Fig. 2. Example protocol in our language

Note that our use of the word “protocol” encompasses not only the “real” protocols — the inputs to the analysis, but also the results of their transformation. Some intuitive assumptions no longer hold for the transformed protocols. Namely, the variables are *shared* among all participants of the protocol. I.e. there may be variables that are defined by one participant and subsequently used by others. It is possible that the adversary has scheduled the participants in such a way that a participant attempts to use a variable whose value has not yet been set. This is the reason to keep track of the initialization of variables. Obviously, in “real” protocols, such sharing does not occur. The transformed protocols are not intended to be actually realized as protocols (where the participants are in physically different locations), they are merely an analysis tool.

A. An example

Consider the Needham-Schroeder key exchange protocol (using secret-key encryption), followed by a transmission of the secret message, encrypted with the new key:

$A \longrightarrow S : A, B, N_A$
$S \longrightarrow A : \mathit{encr}_{K_{AS}}(N_A, B, K_{AB}, \mathit{encr}_{K_{BS}}(K_{AB}, A))$
$A \longrightarrow B : \mathit{encr}_{K_{BS}}(K_{AB}, A)$
$A \longrightarrow B : \mathit{encr}_{K_{AB}}(M)$

Fig. 2 gives the same protocol in the programming language we introduced above. For brevity, we have combined some calculations. The superscripts at some statements are statement labels. They are used in Sec. VII.

We have denoted the lengths of the expected inputs just with $\ell_1, \dots, \ell_4$. Actually,

- $\ell_1(n)$ is the length of the message (A, B, N^A_A) (when the security parameter is n);
- $\ell_2(n)$ is the length of fa;
- $\ell_3(n)$ is the length of fb;
- $\ell_4(n)$ is the length of aM.

The functions $\ell_1, \dots, \ell_4$ can be easily represented using the lengths of constants and the secret message M, the security parameter n, four arithmetic operations and the operations

- mapping n to the length of keys produced by $\mathcal{K}(1^n)$,
- mapping n and the length of a plaintext to the length of the corresponding ciphertext,
- mapping the lengths of two bit-strings to the length of τ applied to these bit-strings.

If we want to consider simultaneously more than one run of the protocol (to find man-in-the-middle attacks) then we include the program for each participant several times — the number of times corresponds to the number of instances that participant is supposed to take part in. Each instance also reflects participant's role in that instance (initiator, responder, etc.) and intended communications partners. Different instances use different variables.

If the adversary may also pose as a legitimate protocol participant E and therefore has some long-term key K_{ES} shared with the server S then the fragment $K_{ES} := \mathbf{gen\_key}; \mathbf{send}\ K_{ES}$ has to occur in the protocol prelude. If the adversary itself can choose the key K_{ES} then just the statement $K_{ES} := \mathbf{receive}$ has to occur in the prelude.

VI. Security definition

$M \in \mathbf{Var}$ is the variable containing the secret message (the approach can be easily extended to several secret variables). The *view of the adversary* is its final state. The protocol is **secure** if for all possible adversaries, the view of the adversary is independent [30] of the secret message, i.e.

$$\{\!|(E[v_M], E[AS]) \,:\, \mathsf{M} \leftarrow D^M_n, E \leftarrow [\![\wp, \mathcal{A}]\!]_n(\mathsf{M})|\!\}_n \approx \{\!|(E'[v_M], E[AS]) \,:\, \mathsf{M} \leftarrow D^M_n, E, E' \leftarrow [\![\wp, \mathcal{A}]\!]_n(\mathsf{M})|\!\}_n \ . \quad (6)$$

Security definitions based on independence are common in language-based security [31], [32]. They mean that everything the adversary deduces about the secret message M also applies for a randomly chosen secret message. Moreover, the adversary could have deduced the same fact even without running the protocol at all, but just by simulating its run, using a randomly generated (with the same distribution as the genuine secret message, its distribution is supposed to be known) bit-string instead of the secret message. All such deducible facts therefore must be trivial in nature.

We also define the *extended view* of the adversary $\mathcal{A}$ as the family of probability distributions occurring to the left of $\approx$ in (6). Clearly, if the extended views of two protocols $\wp$ and $\wp'$ are indistinguishable then they are either both secure or both insecure.

Remark. The security of the protocol also depends on the family of distributions D^M. If this is undesirable then we may universally quantify over it (the analysis given below is correct also for such definition).

VII. Rewriting protocols

A protocol can be made more analyzable by changing it in certain ways. However, we must make sure that all changes are valid, i.e. do not distinguishably change the extended view of the adversary to the protocol.

This section describes a protocol transformation that constructs a protocol $\wp'$ from $\wp$ that makes less use of cryptographic operations. The protocol $\wp$ may not contain $case$-statements, the protocol $\wp'$ may contain them. The protocol $\wp$ is secure iff the protocol $\wp'$ is. We also describe how to replace the protocol $\wp'$ with several protocols $\wp'_1, \wp'_2, \ldots$, such that

- if $\wp'_1, \wp'_2, \ldots$ are secure then $\wp'$ is secure;
- $\wp'_1, \wp'_2, \ldots$ contain no $case$-statements.

The protocols $\wp'_1, \wp'_2, \ldots$ may be analyzed with some information-flow analysis based purely on information theory [33], for example the one presented in the next Section; from their security follows the security of the protocol $\wp$. If some of the protocols $\wp'_1, \wp'_2, \ldots$ still contains significant uses of cryptography, then we may apply the protocol transformation to them again. So, in practice, the protocol transformation consists of several iterations of applying the procedure that turns the initial protocol $\wp$ into a set of protocols $\{\wp'_1, \wp'_2, \ldots\}$ (the set is finite, its cardinality is determined by the structure of $\wp$). At the next iteration, the same procedure is applied to all protocols $\wp'_i$ that are deemed to still contain significant uses of cryptography, etc.

For changing the protocol $\wp$, we want to find the following two items:

- We want to find a key k (actually: a statement $k := \mathbf{gen\_key}$), such that we are able to find all uses of that key (i.e. all uses of the key created at that key generation statement). We must be able to exactly point to all uses of that key; approximation of that set of uses is not allowed.
- We want to find dead code — statements that are never executed. It's permissible to find only a subset of all dead code.

Removing dead code may make the analysis given in Sec. VIII more precise, because some **send**-statements may be removed. The reason to find a key k with the aforementioned properties is to check whether all uses of that key that are necessary in creating the view of the adversary can be replaced with the uses of encryption and decryption oracles, encrypting and decrypting with that key. If that is really the case then the statements of the form $x := encr_k(y)$ may be replaced by $x := encr_k(Z)$, where Z is a new variable whose value is set to $0^{|y|}$. Such replacement does not distinguishably change the view of the adversary because of the security of the encryption system against adaptive CCA. Removing dead code also simplifies the finding of such keys.

The process of defining $\wp'$ has several steps. First we remove all dead code from $\wp$ that we can find through a simple syntactic analysis presented in Sec. VII-A. After that the protocol is "abstractly interpreted". To each variable we assign a term that describes the construction of the value of that variable from atomic values (keys, random numbers, constants, the secret message and the adversary's inputs). We describe this part in Sec. VII-B. The interpretation is necessary for knowing, which keys are used in which statements.

Next a suitable key k is selected. Suitability means that the key is not really necessary for producing the view of the adversary. Instead, only access to black boxes encrypting and decrypting with that key is necessary. We describe this part in Sec. VII-C.

Finally the actual replacement of statements occurs. This is described in Sec. VII-D, the changes presented there define the protocol $\wp'$.

In Sec. VII-E we describe how the protocols $\wp'_1, \wp'_2, \ldots$ are constructed from the protocol $\wp'$.

A. Removing syntactically unreachable code

The order of statements occurring in the protocol puts certain restrictions on the order in which they may be executed. Namely, consider a directed graph $G_\wp$ with the vertex set $L_\wp$ (5) and the following arcs:

- For all $A \in \mathbf{Parts}'$ and $1 \le i \le l_A - 1$ (recall that l_A was the number of statements in the program $\wp(A)$) there is an arc from (A, i) to $(A, i + 1)$.
- For all $A \in \mathbf{Parts}$ there is an arc from $(\mathsf{pre}, l_{\mathsf{pre}})$ to $(A, 1)$.
- If (A, i) is the statement $signal(s)$ and (A', i') is the statement $wait(s)$ (for the same $s \in \mathbf{Sem}$) then there is an arc from (A, i) to (A', i').

Clearly, if there exists a (directed) path from a node (A, i) to a node (A', i'), then in every possible execution, if the statement at (A', i') is executed, then the statement at (A, i) must already have been executed earlier. Therefore, if the graph $G_\wp$ contains (directed) cycles, then no statements in these cycles or following them can be executed — they are dead code. We can remove these statements. In the following we assume that $G_\wp$ is a directed acyclic graph.

B. Interpreting the protocol

We assume that all statements in a protocol have been labeled by elements of some set $\mathbf{Lab}$, such that all statements have different labels. We can identify the sets $\mathbf{Lab}$ and $L_\wp$ (5). In the following we determine how are the values of variables constructed from "atomic" values.

The definition of *terms* $T \in \mathbf{Trm}$ constituting the abstract values of variables is given in Fig. 3. Here $T_1, \ldots, T_m$ are terms, $l \in \mathbf{Lab}$ and $x \in \mathbf{Var}$. We see that the constructors of $\mathbf{Trm}$ more or less correspond to the statements of the programming language. The meaning of terms becomes clearer when we give the abstract interpretation of statements.

$$
\begin{array}{rclcl}
T & ::= & \underline{key}(l) & | & \underline{random}(l) & | & \underline{tuple}^m(T_1, \ldots, T_m) \\
& | & \underline{\pi_i^m}(T_1) & | & \underline{const}(b) & | & \underline{encr}(l, T_1, T_2) \\
& | & \underline{secret} & | & \underline{input}(l) & | & \underline{decr}(T_1, T_2)
\end{array}
$$

Fig. 3. The set of abstract values

As we said before, abstractly interpreting the protocol assigns a term to each variable. Let $\mathsf{I}(x) \in \mathbf{Trm}$ denote the term assigned to the variable x. We let $\mathsf{I}(M) = \underline{secret}$, where $M \in \mathbf{Var}$ was the variable containing the secret value. The rest of the terms $\mathsf{I}(x)$ are defined by examining the statement defining x.

The abstract interpretation of statements defining the value of a variable x defines the term $\mathsf{I}(x)$. It is defined via the label of the statement and the terms assigned to variables at the right hand side of the statement. The definition of $\mathsf{I}(x)$ is given in Fig. 4. We see that a statement label is included in the term iff the result of the statement may depend on something else than the arguments of the statement — namely on random numbers or adversary's input.

$x := \dots$	$\mathsf{I}(x)$ is
$\mathbf{gen\_key}^l$	$\underline{key}(l)$
$\mathbf{random}^l$	$\underline{random}(l)$
$(x_1, \dots, x_m)$	$\underline{tuple}^m(\mathsf{I}(x_1), \dots, \mathsf{I}(x_m))$
$\pi_i^m(y)$	$\underline{\pi_i^m}(\mathsf{I}(y))$
$encr_k(y)^l$	$\underline{encr}(l, \mathsf{I}(k), \mathsf{I}(y))$
$decr_k(y)$	$\underline{decr}(\mathsf{I}(k), \mathsf{I}(y))$
$\mathbf{receive}^l$	$\underline{input}(l)$
$\mathbf{constant}(b)$	$\underline{const}(b)$
y	$\mathsf{I}(y)$

Fig. 4. Definition of the terms $\mathsf{I}(x)$

The abstract interpretation of **send** x does nothing because this statement has no effect on the values of variables.

We can give a semantics to the terms $T \in \mathbf{Trm}$. Let the random choices $\mathbf{r} \in \mathbf{R}$ be fixed. We define $\iota_n^{\mathbf{r}} : \mathbf{Trm} \rightarrow \Sigma_\perp$ as follows (the function $\iota_n^{\mathbf{r}}$ is only defined for terms occurring as the abstract values of variables, as well as for their subterms):

- $\iota_n^{\mathbf{r}}(\underline{input}(l))$ is equal to the value of the variable x defined by the statement $x := \mathbf{receive}^l$. As the random choices are fixed, so is the value of x.
- $\iota_n^{\mathbf{r}}(\underline{key}(l)) = \tau(\mathsf{key}, \mathcal{K}^{\mathbf{r}(l,\mathsf{p})}(1^n))$, i.e. the key is generated with the random coin-tosses fixed as $\mathbf{r}(l, \mathsf{p})$; here l represents the first two arguments of $\mathbf{r}$. We say that the key is *tagged with* key.
- $\iota_n^{\mathbf{r}}(\underline{random}(l))$ is the uniformly generated bit-string of length $p(n)$ (the polynomial p determined the length of random bit-strings) generated with the random coin-tosses $\mathbf{r}(l, \mathsf{p})$ and tagged with random.
- $\iota_n^{\mathbf{r}}(\underline{const}(b)) = \tau(\mathsf{const}, b)$.
- $\iota_n^{\mathbf{r}}(\underline{secret}) = \tau(\mathsf{secret}, \mathbf{r}(M))$.
- $\iota_n^{\mathbf{r}}(\underline{tuple}^m(T_1, \dots, T_m)$ is $\perp$ if any of $\iota_n^{\mathbf{r}}(T_i)$ is $\perp$. Otherwise it is

  $$\tau(\tau(\mathsf{tuple}, m), \tau(\iota_n^{\mathbf{r}}(T_1), \tau(\cdots \tau(\iota_n^{\mathbf{r}}(T_{m-1}), \iota_n^{\mathbf{r}}(T_m))) \cdots)$$

  i.e. the tuple is constructed in the same way as by the execution of the tupling statement.
- $\iota_n^{\mathbf{r}}(\underline{\pi_i^m}(T))$ is $\perp$ if $\iota_n^{\mathbf{r}}(T)$ is $\perp$. Otherwise $\iota_n^{\mathbf{r}}(T)$ is parsed exactly as by the execution of the projection statement. The i-th element of that parse is taken as $\iota_n^{\mathbf{r}}(\underline{\pi_i^m}(T))$. If the parse fails then $\iota_n^{\mathbf{r}}(\underline{\pi_i^m}(T)) = \perp$.
- $\iota_n^{\mathbf{r}}(\underline{decr}(T_1, T_2))$ is $\perp$ if
  - $\iota_n^{\mathbf{r}}(T_1)$ is $\perp$ or not tagged with key;
  - $\iota_n^{\mathbf{r}}(T_2)$ is $\perp$ or not tagged with $\mathsf{ciphertext}$;
  - $\mathcal{D}(1^n, t_1, t_2)$ fails, where $(\_, t_1) = \tau^{-1}(\iota_n^{\mathbf{r}}(T_1))$ and $(\_, t_2) = \tau^{-1}(\iota_n^{\mathbf{r}}(T_2))$.

  Otherwise $\iota_n^{\mathbf{r}}(\underline{decr}(T_1, T_2))$ is equal to the result of that decryption.
- $\iota_n^{\mathbf{r}}(\underline{encr}(l, T_1, T_2))$ is $\perp$ if $\iota_n^{\mathbf{r}}(T_1)$ is $\perp$ or not tagged with key or $\iota_n^{\mathbf{r}}(T_2)$ is $\perp$. Otherwise it is equal to $\tau(\mathsf{ciphertext}, \mathcal{E}^{\mathbf{r}(l,\mathsf{p})}(1^n, t_1, t_2))$ where $(\_, t_1) = \tau^{-1}(\iota_n^{\mathbf{r}}(T_1))$ and $(\_, t_2) = \tau^{-1}(\iota_n^{\mathbf{r}}(T_2))$.

Let $X_n^{\mathbf{r}} \subseteq L_\wp$ denote the set of statements that are actually executed (for the random choices $\mathbf{r}$), i.e. the execution of the protocol is not stopped before reaching that statement and the participant containing that statement does not get stuck at or before that statement.

Theorem 2: Let $x \in \mathbf{Var}$ and let $l \in L_\wp$ be (the label of) the statement defining x. If $l \in X_n^{\mathbf{r}}$ or $x = M$ then

$$[\![\wp, \mathcal{A}]\!]_n^{\mathbf{r}}[v_x] = \iota_n^{\mathbf{r}}(\mathsf{I}(x)) \ . \tag{7}$$

The theorem is proven by induction over the order of executing the statements (determined by $\mathbf{r}$). It mainly consists of comparing the definitions of the execution step and of $\iota_n^{\mathbf{r}}$.

We have rules for deriving the equality or the inequality of two terms (it is also possible that neither can be derived for certain terms). For example, projection and tupling cancel out, as do decryption and encryption (under certain conditions). An example of inequality is that the different keys are unequal. All rules are semantically justified — if we can derive that certain two terms are [in]equal, then there is only insignificant probability that the bit-strings corresponding to these terms are not [in]equal. The probability is taken over protocol runs (i.e. over $\mathbf{r} \in_R \mathbf{R}$).

We denote the derived equality of terms T_1 and T_2 in protocol P by $T_1 \simeq T_2$ and their inequality by $T_1 \not\simeq T_2$. We can also derive that the semantics of a certain term T is almost always[1] $\perp$; we denote this by $\mathsf{stuck}\, T$. Fig. 5 presents the axioms and inference rules defining $\simeq$, $\not\simeq$ and stuck. Most of these rules should be obvious. The rule (nec) says that the type of the value that a bit-string represents can be determined from that bit-string. The rules ($\mathrm{dif_k}$) and ($\mathrm{dif_r}$) say that different random bit-strings are different. The rule ($\mathrm{dif_e}$) is justified by the requirement for the encryption algorithm to almost never return the same bit-string twice. The rules ($\mathrm{inc_p}$), ($\mathrm{inc_{e1}}$) and ($\mathrm{inc_{e2}}$) state that projection and decryption check the types of their inputs.

We say that $T_1 \simeq T_2$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T_1) \neq \perp \neq \iota_n^{\mathbf{r}}(T_2) \wedge \iota_n^{\mathbf{r}}(T_1) \neq \iota_n^{\mathbf{r}}(T_2) \,:\, \mathbf{r} \in_R \mathbf{R}] \tag{8}$$

is negligible in n. Similarly, $T_1 \not\simeq T_2$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T_1) \neq \perp \neq \iota_n^{\mathbf{r}}(T_2) \wedge \iota_n^{\mathbf{r}}(T_1) = \iota_n^{\mathbf{r}}(T_2) \,:\, \mathbf{r} \in_R \mathbf{R}] \tag{9}$$

is negligible in n. I.e. soundness at the statement S means that if S really is executed then the variables whose abstract

[1] the opposite has negligible probability

$$\frac{}{T \simeq T}(\mathrm{id}) \quad \frac{T_1 \simeq T_1' \ \ldots\ T_m \simeq T_m' \quad \underline{C} \text{ is } m\text{-ary constructor of } \mathbf{Trm}}{\underline{C}(T_1,\ldots,T_m) \simeq \underline{C}(T_1',\ldots,T_m')}(\mathrm{ind})$$

$$\frac{}{\underline{\pi_i^m}(\underline{tuple^m}(T_1,\ldots,T_m)) \simeq T_i}(\mathrm{can_t})$$

$$\frac{T_k \simeq T_k'}{\underline{decr}(T_k, \underline{encr}(l, T_k', T)) \simeq T}(\mathrm{can_e})$$

$$\frac{\begin{array}{l}\underline{C}, \underline{C'} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{encr}, \\ \qquad \underline{tuple^2}, \underline{tuple^3}, \underline{tuple^4}, \ldots\} \\ \qquad \underline{C} \neq \underline{C'}\end{array}}{\underline{C}(\ldots) \not\simeq \underline{C'}(\ldots)}(\mathrm{nec})$$

$$\frac{l \neq l'}{\underline{key}(l) \not\simeq \underline{key}(l)}(\mathrm{dif_k}) \quad \frac{l \neq l'}{\underline{random}(l) \not\simeq \underline{random}(l')}(\mathrm{dif_r})$$

$$\frac{T_i \not\simeq T_i'}{\underline{tuple^m}(T_1,\ldots,T_m) \not\simeq \underline{tuple^m}(T_1',\ldots,T_m')}(\mathrm{dif_t})$$

$$\frac{l \neq l'}{\underline{encr}(l, T_1, T_2) \not\simeq \underline{encr}(l', T_3, T_4)}(\mathrm{dif_e})$$

$$\frac{\begin{array}{l}\underline{C} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{encr}, \\ \qquad \underline{tuple^2}, \ldots, \underline{tuple^{m-1}}, \underline{tuple^{m+1}}, \ldots\}\end{array}}{\mathsf{stuck}\, \underline{\pi_i^m}(\underline{C}(\ldots))}(\mathrm{inc_p})$$

$$\frac{\underline{C} \in \{\underline{key}, \underline{random}, \underline{const}, \underline{tuple^2}, \underline{tuple^3}, \ldots\}}{\mathsf{stuck}\, \underline{decr}(T, \underline{C}(\ldots))}(\mathrm{inc_{e1}})$$

$$\frac{\underline{C} \in \{\underline{random}, \underline{const}, \underline{tuple^2}, \underline{tuple^3}, \ldots, \underline{encr}\}}{\mathsf{stuck}\, \underline{decr}(\underline{C}(\ldots), T)}(\mathrm{inc_{e2}})$$

$$\frac{\mathsf{stuck}\, T \quad \underline{C} \text{ is a constructor of } \mathbf{Trm}}{\mathsf{stuck}\, \underline{C}(\ldots, T, \ldots)}(\mathrm{st})$$

Fig. 5. Equality and inequality of terms

values are the terms T_1 and T_2 are [in]equal. Sometimes we can determine the soundness of both $T_1 \simeq T_2$ and $T_1 \not\simeq T_2$ at a certain statement. If this happens and the statement defining the variable whose abstract value is T_1 or T_2 is above S (i.e. there is a path from this statement to S in the graph $G_\wp$) then we have determined that this statement is dead.

We say that $\mathsf{stuck}\, T$ is *sound at the statement* S^l if the probability

$$\mathbf{P}[l \in X_n^{\mathbf{r}} \wedge \iota_n^{\mathbf{r}}(T) \neq \bot \,|\, \mathbf{r} \in_R \mathbf{R}] \qquad (10)$$

is negligible in n. If the statement defining the variable whose abstract value is T is above S then S is dead.

Theorem 3: All rules in Fig. 5 are sound. I.e. if S^l is a statement of the protocol and all the premises of some rule are sound at S^l then the consequent of that rule is also sound at S^l.

See [4] for the proof. Establishing the soundness of each rule is actually quite trivial.

Consider the graph $G_\wp$. We can use the rules in Fig. 5 to determine whether the nodes (i.e. the statements of the protocol) are dead. Fig. 5 has described rules to derive $T_1 \simeq T_2$, $T_1 \not\simeq T_2$ and $\mathsf{stuck}\, T$. Beside them we may also have some extra axioms, in general we have different sets of axioms at different nodes. At the "beginning" of the graph (i.e. at the nodes corresponding to statements that are executed first) we have no extra axioms. Each **check**-statement adds one axiom to the set of axioms at its one node, as well as to all nodes that follow that node. The statement $\mathbf{check}(x = y)$ adds the axiom $\mathsf{I}(x) \simeq \mathsf{I}(y)$. Indeed, if this **check**-statement is actually executed for some choice of random coins $\mathbf{r} \in \mathbf{R}$ then $\iota_n^{\mathbf{r}}(\mathsf{I}(x))$ and $\iota_n^{\mathbf{r}}(\mathsf{I}(y))$ must be equal. Their equality is therefore also necessary for the execution of any statements that follow the statement $\mathbf{check}(x = y)$. I.e. $\mathsf{I}(x) \simeq \mathsf{I}(y)$ is sound at the statement $\mathbf{check}(x = y)$ and all at statements that follow it.

Some of the rules in Fig. 5 can be used to simplify the terms. The rules $(\mathrm{can_t})$ and $(\mathrm{can_e})$ can be applied to the subterms of some term from left to right. When we introduce a special term $\underline{stuck}$ with $\iota_n^{\mathbf{r}}(\underline{stuck}) = \bot$ for all $\mathbf{r}$ and n then we can also use rules (inc_*) and (st) for simplification (denoting stuck terms with $\underline{stuck}$). We have shown that such simplifications are sound in certain sense — Theorem 2 still holds for all but a negligible fraction of possible random choices $\mathbf{r} \in \mathbf{R}$.

Some other derived relationships $T_1 \simeq T_2$ may also be useful for "simplification". In particular, whenever we find that $T_1 \simeq T_2$ holds at a statement defining some variable x with $\mathsf{I}(x) = T_1$ for a term T_1 that contains subterms $\underline{input}(l)$ and term T_2 that does not contain such subterms, we simplify T_1 to T_2.

To summarize, the protocol analysis does the following during the interpretation:

1) Defines $\mathsf{I}(x)$ for all $x \in \mathbf{Var}$ according to Fig. 4. Also simplifies these terms.
2) Annotates each statement S with axioms of the form $\mathsf{I}(x) \simeq \mathsf{I}(y)$ for all statements $\mathbf{check}(x = y)$ that precede S (in the order defined by $G_\wp$).
3) For all statements S attempts to determine that they are dead. For this the analysis has to do one of the following:
   - Derive (using the rules in Fig. 5) $\mathsf{I}(x) \simeq \mathsf{I}(y)$ and $\mathsf{I}(x) \not\simeq \mathsf{I}(y)$ for some $x, y \in \mathbf{Var}$ such that either the definition of x or the definition of y precedes S.
   - Derive $\mathsf{stuck}\, \mathsf{I}(x)$ for some $x \in \mathbf{Var}$ such that the definition of x precedes S.
4) Remove all dead statements from the protocol.

C. Choosing the key

We have to choose a key that can be replaced by black boxes encrypting and decrypting with it. I.e. this key may only be used for encryption and decryption. Choosing a key means choosing some $l \in \mathbf{Lab}$, such that $\underline{key}(l)$ occurs in the abstract values of variables.

The adversary becomes information from the protocol through the statements **send** x, where $x \in \mathbf{Var}_{\mathrm{sent}}$, and also through the participants becoming or not becoming stuck. The abstract values of these variables describe, how the values of these variable are constructed. Therefore the rule for a key $\underline{key}(l)$ to be eligible to be chosen is very simple — if the value of some variable x in any way affects the adversary's view, then $\underline{key}(l)$ may occur in $\mathsf{l}(x)$ only in contexts $\underline{encr}(l', \underline{key}(l), T)$ and $\underline{decr}(\underline{key}(l), T)$. Such variables x are

- the elements of $\mathbf{Var}_{\text{sent}}$;
- the variables occurring in **check**-statements;
- the variables occurring as ciphertexts in decryption statements;
- the variables occurring as keys in encryption or decryption statements (exception: $\underline{key}(l)$ itself may be the abstract value $\mathsf{l}(x)$ of x).
- the variables occurring in the right-hand side of projection statements for which we do not statically know whether the projection succeeds or not (we know it if the outermost constructor of $\mathsf{l}(x)$ is $\underline{tuple}^m$, $\underline{key}$, $\underline{random}$, $\underline{const}$, $\underline{secret}$ or $\underline{encr}$).

If we replace the key $\underline{key}(l)$ with black boxes encrypting and decrypting with it, and using a dummy key whenever the key $\underline{key}(l)$ is used in some other context (i.e. its value is used as a plaintext in encryption or as a component in a tuple) then the values of the variables mentioned above do not change.

Note that having subterms $\underline{input}(l')$ occurring in these terms $\mathsf{l}(x)$ does not invalidate the choice of $\underline{key}(l)$. If the bit-string corresponding to $\underline{input}(l')$ were to contain (the bit-string corresponding to) $\underline{key}(l)$, then $\underline{key}(l)$ itself must have been sent out before and therefore occurs in some other context in some $\mathsf{l}(y)$, where $y \in \mathbf{Var}_{\text{sent}}$. If no term $\mathsf{l}(x)$, where $x \in \mathbf{Var}_{\text{sent}}$, contained $\underline{key}(l)$ in contexts other than $\underline{encr}(l', \underline{key}(l), T)$ and $\underline{decr}(\underline{key}(l), T)$, and the bit-string corresponding to some $\underline{input}(l')$ contained the bit-string corresponding to $\underline{key}(l)$, then, by combining the adversary and the protocol, we could obtain an algorithm that finds the key k from the black boxes $\mathcal{E}(1^n, k, \cdot)$ and $\mathcal{D}(1^n, k, \cdot)$. The existence of such algorithm means that the encryption system is not secure against CCA.

The precise meaning of replacing a key with encrypting and decrypting black boxes is the following. Consider a change to the execution of the protocol we described in Sec. V. Namely, when we compute the values v_x, we do not use the right hand side of the statement defining x, but take $v_x = \iota_n^{\mathbf{r}}(x)$ (where $\mathbf{r} \in_R \mathbf{R}$). By theorem 2 and remarks at the description of simplifying terms, there is only a negligible fraction of random coin-tosses $\mathbf{r}$ for which some value v_x changes. We see now that all values through which anything may be communicated to the adversary can now be computed with only the access to two black boxes respectively encrypting and decrypting with $\iota_n^{\mathbf{r}}(\underline{key}(l))$. Access to the value $\iota_n^{\mathbf{r}}(\underline{key}(l))$ itself is not necessary.

Obviously we are not going to choose again a key that has already been chosen on previous iterations of transforming the protocol.

D. Replacing encryptions

Before we replace the encryptions with $\underline{key}(l)$ with the encryptions of bit-strings of all zeroes, we have to make sure that we do not decrypt any values with $\underline{key}(l)$ that were created by encrypting something with $\underline{key}(l)$ (recall the definition of security against CCA). The key $\underline{key}(l)$ is never sent out, therefore we know exactly, where in the protocol it is used. Let $x_1 := encr_{k_1}(y_1), \ldots, x_n := encr_{k_n}(y_n)$ be all such encryption statements in the protocol, that $\mathsf{l}(k_1) \simeq \cdots \simeq \mathsf{l}(k_n) \simeq \underline{key}(l)$. If $w := decr_{k'}(z)$ is a decryption statement in the protocol such that $\mathsf{l}(k') \simeq \underline{key}(l)$, then we replace it by

$$\begin{array}{l} w := case\ z\ of \\ \qquad\qquad x_1 \quad \rightarrow y_1 \\ \qquad\qquad \ldots\ldots\ldots\ldots\ldots\ldots \\ \qquad\qquad x_n \quad \rightarrow y_n \\ \qquad\qquad else \rightarrow decr_{k'}(z) \end{array}$$

Clearly, such replacement does not change the execution of the protocol at all. This *case*-statement is somewhat different than (4) — it has an *else*-clause. But as $x_1, \ldots, x_n$ were all ciphertexts produced by encrypting with $\underline{key}(l)$ and as the used encryption system has ciphertext integrity, the expression $decr_{k'}(z)$ in the *else*-clause almost always fails (i.e. the value of z is not a valid ciphertext). Therefore we may remove the *else*-clause and get the *case*-statement (4).

The key $\underline{key}(l)$ now occurs in the protocol $\wp$ only in contexts allowed by the definition of security against CCA. We can therefore replace the statements $x_i := encr_{k_i}(y_i)$ ($i \in \{1, \ldots, n\}$), defined as above, with $x_i := encr_{k_i}(Z_{y_i})$, where Z_{y_i} is a variable initialised to $\mathbf{constant}(0^{|y_i|})$ in the protocol prelude. The resulting protocol is the protocol $\wp'$.

E. Removing case-statements

If the number of branches in some *case*-statement is zero then this statement is dead code. We remove it and everything succeeding it. In the following we assume that each *case*-statement has at least one branch.

Let $\mathsf{S}_1, \ldots, \mathsf{S}_k$ be the *case*-statements in the protocol $\wp'$. Let the statement S_i have n_i cases. The number of protocols $\wp'_1, \wp'_2, \ldots$ will be $n_1 \cdot n_2 \cdots n_k$; they are defined in the following way.

For all $i \in \{1, \ldots, n\}$ choose $c_i \in \{1 \ldots, n_i\}$; each of the protocols $\wp'_1, \wp'_2, \ldots$ will correspond to one of such possible choices. Let $s_1, \ldots, s_n \in \mathbf{Sem}$ be semaphores not occurring in the protocol $\wp'$. To obtain a new protocol, we change the *case*-statement S_i (which we assume to be of the form (4)) in the following way:

- Add $signal(s_i)$ immediately after the statement defining x_{c_i}.
- Replace S_i with the following fragment:

$$\begin{array}{l} wait(s_i) \\ \mathbf{check}(z = x_{c_i}) \\ w := y_{c_i} \end{array}$$

Such change is applied to all *case*-statements. The resulting protocol has no *case*-statements. Basically, going from $\wp'$ to

$\wp_1', \wp_2', \ldots$ just means analyzing all branches of the *case*-statements separately.

We are analyzing the branches separately, therefore we may wonder whether we are missing some implicit flows of information here. In other words, is it possible that all n programs that we get after transforming a *case*-statement are secure, but the original program is not secure? Fortunately, this is not possible. In our programming language we have no means to let the value of the secret M influence, which of the branches will be taken. Indeed, there was no *case*-statement in our original programming language. Therefore the choice of a branch only depends on the adversary's actions. If the adversary does not know M before one of the branches is chosen then the chosen branch does not depend on M and hence gives no information about M to the adversary.

VIII. A SIMPLE INFORMATION-FLOW ANALYSIS

We could use some program analysis for secure information flow for trying to find out whether the confidentiality of the secret message M is preserved. Such analyses require specifying two subsets of $\mathbf{Var}$ — the sets of private input variables (in our case $\{M\}$) and public output variables (in our case $\mathbf{Var}_{\mathrm{sent}}$). The analysis checks that the final values of the public output variables do not depend on the initial values of the private input variables.

We will now present a very simple information flow analysis, similar to [34], [35]. If some participant of the protocol contains a statement of the form $x := E(x_1, \ldots, x_m)$, where E is any expression and $x_1, \ldots, x_m$ are all variables occurring in it, then we say that there is *information flow* from the variable x_i to the variable x and write $x_i \Rightarrow x$. The protocol P is deemed secure if $M \overset{*}{\Rightarrow} y$ holds for no y satisfying some of the following conditions:

- $y \in \mathbf{Var}_{\mathrm{sent}}$;
- y occurs in a **check**-statement;
- y occurs as an argument to a projection (π) statement;
- y occurs as a ciphertext is a decryption statement;
- y occurs as a key in an encryption or decryption statement.

(here $\overset{*}{\Rightarrow}$ denotes the reflexive transitive closure of $\Rightarrow$). Otherwise the protocol is deemed potentially insecure.

We use that analysis to analyse the set of protocols that we construct in the next section from the protocol that is given to us. It is to conservative to give any interesting results for the original protocol itself. The correctness of the analysis can be easily proved (similarly to [36]).

We are going to prove the following result about the described analysis, similar in vein to [36], stating that if a protocol is secure then two instances of that protocol which differ only in the value of the secret variable M execute in lock-step, giving the same values to all public parts of the execution context.

Let E_1, E_2 be two execution states. We say that E_1, E_2 are *similar* (denote $E_1 \sim E_2$), if their following components are equal:

- sm, rm, AS, C;
- $\mathsf{P}(A)$ for all $A \in \mathbf{Parts}'$;
- The flags r_A (for $A \in \mathbf{Parts}'$), i_x (for $x \in \mathbf{Var}$), p_s (for $s \in \mathbf{Sem}$);
- v_x for all $x \in \mathbf{Var}$ such that $\neg(M \overset{*}{\Rightarrow} x)$;
- v_x for all $x \in \mathbf{Var}$ such that $E_1[v_x]$ is tagged with key.

Let $\mathsf{M}, \mathsf{M}' \in \Sigma$ and let $\mathbf{r}, \mathbf{r}' \in \mathbf{R}$ be two possible results of random coin-tosses, such that $\mathbf{r}(M) = \mathsf{M}$, $\mathbf{r}'(M) = \mathsf{M}'$ and $\mathbf{r}$ and $\mathbf{r}'$ are equal on all other points.

Theorem 4: If $E_1 \sim E_2$ and $E_1 \xrightarrow{\mathbf{r}} E_1'$ and $E_2 \xrightarrow{\mathbf{r}'} E_2'$ then $E_1' \sim E_2'$.

The proof is similar to [36]. See [4].

The initial execution states are similar for the secret values m and m'. By the last theorem, the final states are similar also, i.e. they agree on the value AS. The final state of the adversary thus does not depend on the secret value.

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
M	$\underline{secret}$
K_{AS}	$\underline{key}(1)$
K_{BS}	$\underline{key}(2)$
A	$\underline{const}(\text{“A”})$
B	$\underline{const}(\text{“B”})$
N_A^A	$\underline{random}(3)$
m_2^A	$\underline{input}(4)$
dm_2^A	$\underline{decr}(\underline{key}(1), \underline{input}(4))$
$\pi_1^4(dm_2^A)$	$\underline{\pi_1^4}(\underline{decr}(\underline{key}(1), \underline{input}(4)))$
$\pi_4^4(dm_2^A)$	$\underline{\pi_4^4}(\underline{decr}(\underline{key}(1), \underline{input}(4)))$
K_{AB}^A	$\underline{\pi_3^4}(\underline{decr}(\underline{key}(1), \underline{input}(4)))$
aM	$\underline{encr}(5, \underline{\pi_3^4}(\underline{decr}(\underline{key}(1), \underline{input}(4))), \underline{secret})$
m_1	$\underline{input}(6)$
N_A^S	$\underline{\pi_3^3}(\underline{input}(6))$
K_{AB}	$\underline{key}(7)$
dfb	$\underline{tuple^2}(\underline{key}(7), \underline{const}(\text{“A”}))$
fb	$\underline{encr}(8, \underline{key}(2), \underline{tuple^2}(\underline{key}(7), \underline{const}(\text{“A”})))$
dfa	$\underline{tuple^4}(\underline{\pi_3^3}(\underline{input}(6)), \underline{const}(\text{“B”}), \underline{key}(7),$ $\underline{encr}(8, \underline{key}(2), \underline{tuple^2}(\underline{key}(7), \underline{const}(\text{“A”}))))$
fa	$\underline{encr}(9, \underline{key}(1),$ $\underline{tuple^4}(\underline{\pi_3^3}(\underline{input}(6)), \underline{const}(\text{“B”}), \underline{key}(7),$ $\underline{encr}(8, \underline{key}(2), \underline{tuple^2}(\underline{key}(7), \underline{const}(\text{“A”})))))$
m_3	$\underline{input}(10)$
dm_3	$\underline{decr}(\underline{key}(2), \underline{input}(10))$
K_{AB}^B	$\underline{\pi_1^2}(\underline{decr}(\underline{key}(2), \underline{input}(10)))$
m_4	$\underline{input}(11)$
bM	$\underline{decr}(\underline{\pi_1^2}(\underline{decr}(\underline{key}(2), \underline{input}(10))), \underline{input}(11))$

Fig. 6. Interpretation of protocol in Fig. 2

IX. AN EXAMPLE

We continue the example we presented in Sec. V.

As the protocol given in Fig. 2 is a “real” protocol, it does not contain any dead code. Consider the abstract interpretation of the protocol, it is given in Fig. 6.

The axioms added by **check**-statements do not allow us to simplify any term. The next step of the protocol transformation

Protocol prelude	the program for B
$K_{AS} := \mathbf{gen\_key}^{(1)}$	$m_3 := \mathbf{receive}^{\ell_3\,(10)}$
$K_{BS} := \mathbf{gen\_key}^{(2)}$	$dm_3 := decr_{K_{BS}}(m_3)$
$A := \mathbf{constant}(\text{“A”})$	$K^B_{AB} := \pi^2_1(m_3)$
$B := \mathbf{constant}(\text{“B”})$	$m_4 := \mathbf{receive}^{\ell_4\,(11)}$
$Z_{dfa} := \mathbf{constant}(0^{\ell_5(n)})$	$bM := decr_{K^B_{AB}}(m_4)$

the program for A	the program for S
$N^A_A := \mathbf{random}^{(3)}$	
$\mathbf{send}\ (A, B, N^A_A)$	$m_1 := \mathbf{receive}^{\ell_1\,(6)}$
$m^A_2 := \mathbf{receive}^{\ell_2\,(4)}$	$N^S_A := \pi^3_3(m_1)$
$dm^A_2 := case\ m^A_2\ of$	$K_{AB} := \mathbf{gen\_key}^{(7)}$
$fa \to dfa$	$dfb := (K_{AB}, A)$
$\mathbf{check}(N^A_A = \pi^4_1(dm^A_2))$	$fb := encr_{K_{BS}}(dfb)^{(8)}$
$\mathbf{send}\ \pi^4_4(dm^A_2)$	$dfa := (N^S_A, B, K_{AB}, fb)$
$K^A_{AB} := \pi^4_3(dm^A_2)$	$fa := encr_{K_{AS}}(Z_{dfa})^{(9)}$
$aM := encr_{K^A_{AB}}(M)^{(5)}$	$\mathbf{send}\ fa$
$\mathbf{send}\ aM$	

Fig. 7. Example protocol, first iteration, replacing encryptions

Protocol prelude	the program for B
$K_{AS} := \mathbf{gen\_key}^{(1)}$	$m_3 := \mathbf{receive}^{\ell_3\,(10)}$
$K_{BS} := \mathbf{gen\_key}^{(2)}$	$dm_3 := decr_{K_{BS}}(m_3)$
$A := \mathbf{constant}(\text{“A”})$	$K^B_{AB} := \pi^2_1(m_3)$
$B := \mathbf{constant}(\text{“B”})$	$m_4 := \mathbf{receive}^{\ell_4\,(11)}$
$Z_{dfa} := \mathbf{constant}(0^{\ell_5(n)})$	$bM := decr_{K^B_{AB}}(m_4)$

the program for A	the program for S
$N^A_A := \mathbf{random}^{(3)}$	
$\mathbf{send}\ (A, B, N^A_A)$	$m_1 := \mathbf{receive}^{\ell_1\,(6)}$
$m^A_2 := \mathbf{receive}^{\ell_2\,(4)}$	$N^S_A := \pi^3_3(m_1)$
$wait(s_1)$	$K_{AB} := \mathbf{gen\_key}^{(7)}$
$\mathbf{check}(m^A_2 = fa)$	$dfb := (K_{AB}, A)$
$dm^A_2 := dfa$	$fb := encr_{K_{BS}}(dfb)^{(8)}$
$\mathbf{check}(N^A_A = \pi^4_1(dm^A_2))$	$dfa := (N^S_A, B, K_{AB}, fb)$
$\mathbf{send}\ \pi^4_4(dm^A_2)$	$fa := encr_{K_{AS}}(Z_{dfa})^{(9)}$
$K^A_{AB} := \pi^4_3(dm^A_2)$	$signal(s_1)$
$aM := encr_{K^A_{AB}}(M)^{(5)}$	$\mathbf{send}\ fa$
$\mathbf{send}\ aM$	

Fig. 8. Example protocol, first iteration, removing *case*-statements

is choosing the key as in Sec. VII-C. There are three keys in the protocol — $\underline{key}(1)$, $\underline{key}(2)$ and $\underline{key}(7)$. The keys $\underline{key}(1)$ and $\underline{key}(2)$ satisfy the requirements put forth in Sec. VII-C, the key $\underline{key}(7)$ does not (consider $\mathsf{I}(fa)$, we have $fa \in \mathbf{Var}_{\mathrm{sent}}$). We were required to choose one key (although the transformation could be modified to handle several suitable keys at once), so let us choose $\underline{key}(1)$.

We have to locate all uses of $\underline{key}(1)$. There is just one variable whose abstract value is $\underline{key}(1)$, it is K_{AS}. It is used at the definition of fa for encrypting and dm^A_2 for decrypting. We replace that decryption with a *case*-statement. After also replacing the encryption statement (and introducing the variable Z_{dfa}) we get the protocol in Fig. 7. We denote the length of dfa by ℓ_5.

There is just one *case*-statement in the protocol in Fig. 7, it has a single branch. Therefore, when we remove the *case*-statements we get just one new protocol. Let $s_1 \in \mathbf{Sem}$ be a semaphore. The new protocol is given in Fig. 8.

We continue with the next iteration of the transformation. There still is no syntactically locatable dead code. The results of the abstract interpretation of the protocol are given in Fig. 9 (only the abstract values that have changed from the previous iteration are given). This time only the key $\underline{key}(2)$ satisfies the requirements put forth in Sec. VII-C (actually $\underline{key}(1)$ satisfies them, too, but replacing the statements where $\underline{key}(1)$ is used for encryption would not change the protocol at all).

We have to locate all uses of $\underline{key}(2)$. There is just one variable whose abstract value is $\underline{key}(2)$, it is K_{BS}. It is used at the definition of fb for encrypting and dm_3 for decrypting. After making the same changes (replacing the decryption with a *case*-statement, replacing the encryption and removing the *case*-statement) we get the protocol in Fig. 10. Let ℓ_6 denote the length of the value of the variable dfb.

The analysis given in Sec. VIII still does not consider the protocol secure because $M \overset{*}{\Rightarrow} aM$ and $aM \in \mathbf{Var}_{\mathrm{sent}}$. Let us do another iteration of the protocol transformation. The abstract values of the variables are given in Fig. 11. This

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
Z_{dfa}	$\underline{const}(0^{\ell_5(n)})$
dm^A_2	$\underline{tuple}^4(\underline{\pi}^3_3(\underline{input}(6)), \underline{const}(\text{“B”}), \underline{key}(7), \underline{encr}(8, \underline{key}(2), \underline{tuple}^2(\underline{key}(7), \underline{const}(\text{“A”}))))$
$\pi^4_1(dm^A_2)$	$\underline{\pi}^3_3(\underline{input}(6))$
$\pi^4_4(dm^A_2)$	$\underline{encr}(8, \underline{key}(2), \underline{tuple}^2(\underline{key}(7), \underline{const}(\text{“A”})))$
K^A_{AB}	$\underline{key}(7)$
aM	$\underline{encr}(5, \underline{key}(7), \underline{secret})$
fa	$\underline{encr}(9, \underline{key}(1), \underline{const}(0^{\ell_5(n)}))$

Fig. 9. Interpretation of protocol in Fig. 8 (changes only)

Protocol prelude	the program for B
	$m_3 := \mathbf{receive}^{\ell_3\,(10)}$
$K_{AS} := \mathbf{gen\_key}^{(1)}$	$wait(s_2)$
$K_{BS} := \mathbf{gen\_key}^{(2)}$	$\mathbf{check}(m_3 = fb)$
$A := \mathbf{constant}(\text{“A”})$	$dm_3 := dfb$
$B := \mathbf{constant}(\text{“B”})$	$K^B_{AB} := \pi^2_1(m_3)$
$Z_{dfa} := \mathbf{constant}(0^{\ell_5(n)})$	$m_4 := \mathbf{receive}^{\ell_4\,(11)}$
$Z_{dfb} := \mathbf{constant}(0^{\ell_6(n)})$	$bM := decr_{K^B_{AB}}(m_4)$

the program for A	the program for S
$N^A_A := \mathbf{random}^{(3)}$	
$\mathbf{send}\ (A, B, N^A_A)$	$m_1 := \mathbf{receive}^{\ell_1\,(6)}$
$m^A_2 := \mathbf{receive}^{\ell_2\,(4)}$	$N^S_A := \pi^3_3(m_1)$
$wait(s_1)$	$K_{AB} := \mathbf{gen\_key}^{(7)}$
$\mathbf{check}(m^A_2 = fa)$	$dfb := (K_{AB}, A)$
$dm^A_2 := dfa$	$fb := encr_{K_{BS}}(Z_{dfb})^{(8)}$
$\mathbf{check}(N^A_A = \pi^4_1(dm^A_2))$	$signal(s_2)$
$\mathbf{send}\ \pi^4_4(dm^A_2)$	$dfa := (N^S_A, B, K_{AB}, fb)$
$K^A_{AB} := \pi^4_3(dm^A_2)$	$fa := encr_{K_{AS}}(Z_{dfa})^{(9)}$
$aM := encr_{K^A_{AB}}(M)^{(5)}$	$signal(s_1)$
$\mathbf{send}\ aM$	$\mathbf{send}\ fa$

Fig. 10. Example protocol after second iteration

$x \in \mathbf{Var}$	$\mathsf{I}(x)$
Z_{dfb}	$\underline{const}(0^{\ell_6(n)})$
dm_2^A	$\underline{tuple}^4(\pi_3^3(\underline{input}(6)), \underline{const}(\text{"B"}), \underline{key}(7),$ $\underline{encr}(8, \underline{key}(2), \underline{const}(0^{\ell_6(n)})))$
$\pi_4^4(dm_2^A)$	$\underline{encr}(8, \underline{key}(2), \underline{const}(0^{\ell_6(n)}))$
fb	$\underline{encr}(8, \underline{key}(2), \underline{const}(0^{\ell_6(n)}))$
dfa	$\underline{tuple}^4(\pi_3^3(\underline{input}(6)), \underline{const}(\text{"B"}), \underline{key}(7),$ $\underline{encr}(8, \underline{key}(2), \underline{const}(0^{\ell_6(n)})))$
dm_3	$\underline{tuple}^2(\underline{key}(7), \underline{const}(\text{"A"}))$
K_{AB}^B	$\underline{key}(7)$
bM	$\underline{decr}(\underline{key}(7), \underline{input}(11))$

Fig. 11. Interpretation of protocol in Fig. 10 (changes only)

time the key $\underline{key}(7)$ also satisfies the requirements put forth in Sec. VII-C. Indeed,

- The set $\mathbf{Var}_{\text{sent}}$ is $\{A, B, N_A^A, \pi_4^4(dm_2^A), aM, fa\}$. Of these variables, only the abstract value of aM contains $\underline{key}(7)$ and it occurs there as the encryption key.
- The set of variables occurring in **check**-statements is $\{m_2^A, fa, N_A^A, \pi_1^4(dm_2^A),\ m_3, fb\}$. None of the abstract values of these variables contains $\underline{key}(7)$.
- The set of variables occurring as ciphertexts in decryption statements is empty as there are no decryption statements.
- The set of variables that are used as keys in encryption and decryption statements is $\{K_{AS}, K_{BS}, K_{AB}, K_{AB}^A, K_{AB}^B\}$. The abstract value of the last three variables is $\underline{key}(7)$, this is permitted.
- The set of variables that occur in projection statements is $\{dm_2^A, m_1, m_3\}$. Of these variables, the abstract value of dm_2^A contains $\underline{key}(7)$. But as the outermost constructor of $\mathsf{I}(dm_2^A)$ is $\underline{tuple}^4$, it is known statically whether some projection succeeds or fails (in our case, all of them succeed, because the projection operator is π_i^4).

We can now replace the decryptions with $\underline{key}(7)$ with *case*-statements, the encryption with $\underline{key}(7)$ with the encryption of Z_M (let ℓ_7 be the length of M) and remove the only *case*-statement that was introduced (let $s_3 \in \mathbf{Sem}$ be a semaphore), giving the protocol in Fig. 12. The analysis given in Sec. VIII will consider the protocol in Fig. 12 secure because the variable M is used only in defining bM and that variable is not sent out over the network, nor does the termination of execution depend on its value.

Protocol prelude	the program for B
$K_{AS} := \mathbf{gen\_key}^{(1)}$ $K_{BS} := \mathbf{gen\_key}^{(2)}$ $A := \mathbf{constant}(\text{"A"})$ $B := \mathbf{constant}(\text{"B"})$ $Z_{dfa} := \mathbf{constant}(0^{\ell_5(n)})$ $Z_{dfb} := \mathbf{constant}(0^{\ell_6(n)})$ $Z_M := \mathbf{constant}(0^{\ell_7(n)})$	$m_3 := \mathbf{receive}^{\ell_3\,(10)}$ $wait(s_2)$ $\mathbf{check}(m_3 = fb)$ $dm_3 := dfb$ $K_{AB}^B := \pi_1^2(m_3)$ $m_4 := \mathbf{receive}^{\ell_4\,(11)}$ $wait(s_3)$ $\mathbf{check}(m_4 = aM)$ $bM := M$

the program for A	the program for S
$N_A^A := \mathbf{random}^{(3)}$ $\mathbf{send}\ (A, B, N_A^A)$ $m_2^A := \mathbf{receive}^{\ell_2\,(4)}$ $wait(s_1)$ $\mathbf{check}(m_2^A = fa)$ $dm_2^A := dfa$ $\mathbf{check}(N_A^A = \pi_1^4(dm_2^A))$ $\mathbf{send}\ \pi_4^4(dm_2^A)$ $K_{AB}^A := \pi_3^4(dm_2^A)$ $aM := encr_{K_{AB}^A}(Z_M)^{(5)}$ $signal(s_3)$ $\mathbf{send}\ aM$	$m_1 := \mathbf{receive}^{\ell_1\,(6)}$ $N_A^S := \pi_3^3(m_1)$ $K_{AB} := \mathbf{gen\_key}^{(7)}$ $dfb := (K_{AB}, A)$ $fb := encr_{K_{BS}}(Z_{dfb})^{(8)}$ $signal(s_2)$ $dfa := (N_A^S, B, K_{AB}, fb)$ $fa := encr_{K_{AS}}(Z_{dfa})^{(9)}$ $signal(s_1)$ $\mathbf{send}\ fa$

Fig. 12. Example protocol after third iteration

There are several directions in which the presented work could be extended. We could consider richer program languages for expressing the protocol parties. This would also mean that the interpretation presented in Sec. VII-B and choice of keys presented Sec. VII-C becomes more complex. It is also possible that we can no longer always replace the *case*-statement with a number of equality checks and assignments. Instead, an analysis similar to Sec. VIII may be necessary to find out whether it is allowed. For *case*-statements that we cannot replace, we have to introduce new constructs to the set $\mathbf{Trm}$. These new constructs will then bring more rules to definitions of $\simeq$ and $\not\simeq$.

We could consider integrity properties. Our machinery for protocol interpretation may be quite helpful in determining whether a certain state is reachable or not.

We could try to add more different cryptographic primitives. We can probably add some cryptographic primitive rather easily if

1) its security definition requires it to be indistinguishable from some "ideal system".
2) the workings of this "ideal system" are easily expressible in our program language.

Possible candidates are public-key encryption, digital signatures, message authentication codes etc. We may also consider different security definitions of cryptographic primitives, for example pseudorandom permutations as symmetric encryption.

X. Conclusion

We have presented an analysis for checking whether cryptographic protocols preserve the confidentiality of secret messages. The analysis works by first applying the transformation described in Sec. VII to the protocol, resulting in one or more transformed protocols. The security of all these protocols is checked by the analysis described in Sec. VIII. The original protocol is deemed secure if all transformed protocols were secure. Our analysis is correct with respect to computational semantics and (complexity-theoretical) definition of confidentiality.

Acknowledgments

We are thankful to the anonymous referees of 2004 IEEE Security and Privacy conference and to Birgit Pfitzmann for their comments. The work was supported by Estonian Science Foundation, grant #5279.

REFERENCES

[1] M. Abadi and P. Rogaway, "Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption)," in *International Conference IFIP TCS 2000*, ser. LNCS, J. van Leeuwen, O. Watanabe, M. Hagiya, P. Mosses, and T. Ito, Eds., vol. 1872. Sendai, Japan: Springer-Verlag, Aug. 2000, pp. 3–22.

[2] M. Abadi and J. Jürjens, "Formal Eavesdropping and Its Computational Interpretation," in *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001*, ser. LNCS, N. Kobayashi and B. Pierce, Eds., vol. 2215. Sendai, Japan: Springer-Verlag, Sept. 2001, pp. 82–94.

[3] E. Hermann, "Krüptograafiliste protokollide turvalisuse analüüs (Security Analysis of Cryptographic Protocols)," Diploma thesis, Tartu University, May 2003, in Estonian with English summary.

[4] P. Laud, "Symmetric encryption in automatic analyses for confidentiality against active adversaries," Cybernetica, Tech. Rep. IT-AR-O-030-040227, 27 Feb. 2004.

[5] ——, "Computationally Secure Information Flow," Ph.D. dissertation, Universität des Saarlandes, 2002.

[6] ——, "Handling Encryption in Analyses for Secure Information Flow," in *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003*, ser. LNCS, P. Degano, Ed., vol. 2618. Warsaw, Poland: Springer-Verlag, Apr. 2003, pp. 159–173.

[7] B. Pfitzmann, M. Schunter, and M. Waidner, "Cryptographic Security of Reactive Systems," in *Workshop on Secure Architectures and Information Flow*, ser. Electronic Notes in Theoretical Computer Science, S. Schneider and P. Ryan, Eds., vol. 32. Royal Holloway, University of London: Elsevier Science, 2000.

[8] B. Pfitzmann and M. Waidner, "Composition and integrity preservation of secure reactive systems," in *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security*. Athens, Greece: ACM Press, Nov. 2000, pp. 245–254.

[9] ——, "A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission," in *2001 IEEE Symposium on Security and Privacy*. Oakland, California: IEEE Computer Society Press, May 2001, pp. 184–200.

[10] M. Backes, "Cryptographically Sound Analysis of Security Protocols," Ph.D. dissertation, Universität des Saarlandes, 2002.

[11] M. Backes and C. Jacobi II, "Cryptographically Sound and Machine-Assisted Verification of Security Protocols," in *STACS 2003, 20th Annual Symposium on Theoretical Aspects of Computer Science*, ser. LNCS, H. Alt and M. Habib, Eds., vol. 2607. Berlin, Germany: Springer-Verlag, Feb. 2003, pp. 675–686.

[12] M. Backes, B. Pfitzmann, and M. Waidner, "A Universally Composable Cryptographic Library," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*. Washington, DC: ACM Press, Oct. 2003.

[13] B. Blanchet, "Automatic Verification of Cryptographic Protocols: A Logic Programming Approach (invited talk)," in *5th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'03)*. Uppsala, Sweden: ACM Press, Aug. 2003.

[14] M. Backes, B. Pfitzmann, and M. Waidner, "Symmetric authentication within a simulatable cryptographic library," in *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, ser. LNCS, E. Snekkenes and D. Gollmann, Eds., vol. 2808. Gjøvik, Norway: Springer-Verlag, Oct. 2003, pp. 271–290.

[15] M. Backes and B. Pfitzmann, "Symmetric Encryption in a Simulatable Dolev-Yao Style Cryptographic Library," Cryptology ePrint Archive, Report 2004/059, 2004, http://eprint.iacr.org/.

[16] R. Canetti, "A unified framework for analyzing security of protocols," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 8, no. 16, 2001.

[17] R. Canetti and M. Fischlin, "Universally Composable Commitments," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, ser. LNCS, J. Kilian, Ed., vol. 2139. Santa Barbara, California: Springer-Verlag, Aug. 2001, pp. 19–40.

[18] R. Canetti and H. Krawczyk, "Universally Composable Notions of Key Exchange and Secure Channels," in *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Application of Cryptographic Techniques*, ser. LNCS, L. R. Knudsen, Ed., vol. 2332. Amsterdam, The Netherlands: Springer-Verlag, Apr. 2002, pp. 337–351.

[19] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing*. Montreal, Quebec: ACM, May 2002, pp. 494–503.

[20] J. Herzog, "Computational Soundness of Formal Adversaries," Master's thesis, Massachusetts Institute of Technology, Sept. 2002.

[21] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among Notions of Security for Public-Key Encryption Schemes," in *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference*, ser. LNCS, H. Krawczyk, Ed., vol. 1462. Santa Barbara, California: Springer-Verlag, Aug. 1998, pp. 26–45.

[22] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security*. Fairfax, Virginia: ACM Press, Nov. 1993, pp. 62–73.

[23] J. Herzog, M. Liskov, and S. Micali, "Plaintext Awareness via Key Registration," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference*, ser. LNCS, D. Boneh, Ed., vol. 2729. Santa Barbara, California: Springer-Verlag, Aug. 2003, pp. 548–564.

[24] M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," in *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security*, ser. LNCS, T. Okamoto, Ed., vol. 1976. Kyoto, Japan: Springer-Verlag, Dec. 2000, pp. 531–545.

[25] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption," in *38th Annual Symposium on Foundations of Computer Science*. Miami Beach, Florida: IEEE Computer Society Press, Oct. 1997, pp. 394–403.

[26] M. Naor and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," in *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*. Baltimore, Maryland: ACM Press, May 1990, pp. 427–437.

[27] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference*, ser. LNCS, J. Feigenbaum, Ed., vol. 576. Santa Barbara, California: Springer-Verlag, Aug. 1991, pp. 433–444.

[28] F. Thayer, J. Herzog, and J. Guttman, "Strand Spaces: Proving Security Protocols Correct," *Journal of Computer Security*, vol. 7, no. 2/3, pp. 191–230, 1999.

[29] M. Bellare and P. Rogaway, "Provably secure session key distribution - The three party case," in *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*. Las Vegas, Nevada: ACM Press, June 1995, pp. 57–66.

[30] A. Yao, "Theory and applications of trapdoor functions (extended abstract)," in *23rd Annual Symposium on Foundations of Computer Science*. Chicago, Illinois: IEEE Computer Society Press, Nov. 1982, pp. 80–91.

[31] K. Leino and R. Joshi, "A Semantic Approach to Secure Information Flow," in *Mathematics of Program Construction, MPC '98*, ser. LNCS, J. Jeuring, Ed., vol. 1422. Marstrand, Sweden: Springer-Verlag, June 1998, pp. 254–271.

[32] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov, "A Probabilistic Poly-Time Framework for Protocol Analysis," in *CCS '98, Proceedings of the 5th ACM Conference on Computer and Communications Security*. San Francisco, California: ACM Press, Nov. 1998, pp. 112–121.

[33] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, 1948.

[34] D. Denning, "A Lattice Model of Secure Information Flow," *Communications of the ACM*, vol. 19, no. 5, pp. 236–243, 1976.

[35] D. Denning and P. Denning, "Certification of Programs for Secure Information Flow," *Communications of the ACM*, vol. 20, no. 7, pp. 504–513, 1977.

[36] D. Volpano, G. Smith, and C. Irvine, "A Sound Type System for Secure Flow Analysis," *Journal of Computer Security*, vol. 4, no. 2,3, pp. 167–187, 1996.