

Formal Methods and Cryptography

Short notes, WS 2009/10

Last Update: February 19, 2010

Important note: These notes are not supposed to be self-contained. Instead, they are intended as a reminder about which topics were discussed in the lecture. If you find mistakes in these notes, please send them to unruh@mmci.uni-saarland.de.

Contents

1	Basic Notation	2
2	Example: Protocol analysis	3
2.1	Cryptographer's analysis	3
2.2	Analysis in the Dolev-Yao model	4
3	The Dolev-Yao approach: counterexamples	4
4	Inference Rules and Induction	8
5	Symbolic models	9
6	Computational soundness: The passive case	10
7	Computational soundness: The active case	13
7.1	Protocol model	13
7.2	Symbolic execution	14
7.3	Computational execution	15
7.4	Simulators and the hybrid execution	17
8	Computational soundness of public key encryption	18
8.1	The model	18
8.2	The simulator	20
8.3	Properties of the simulator	21
9	Automated game-based proofs	22
9.1	The calculus used in CryptoVerif	22
9.2	Game transformations	24
9.2.1	Simplification	24

9.2.2	Crypto transformations	25
10	Computational soundness of zero-knowledge proofs	26
10.1	Symbolic model	27
10.2	Computational implementation	28
10.3	ZK-safe protocols	30
10.4	Computational soundness proof	31

1 Basic Notation

By \mathbb{N} we denote the natural numbers including 0. We call a function f *negligible* if for every positive polynomial p , we have that $|f(n)| \leq \frac{1}{p(n)}$ for sufficiently large n . We call a function f *noticeable* if there is a positive polynomial p such that for sufficiently large n , we have $f(n) \geq \frac{1}{p(n)}$. Note that no function can be both negligible and noticeable, but there are functions that are neither noticeable nor negligible (e.g., $f(2n) := 0$, $f(2n + 1) := 1$). We call a function *non-negligible* if it is not negligible. We call a function f *overwhelming* if $f \geq 1 - g$ for some negligible g . We write 2^S for the powerset of S , i.e., $2^S = \{M : M \subseteq S\}$. Given a function f , we write $f(x := y)$ for the function f' satisfying $f'(x) = y$ and $f'(x') = f(x')$ for all $x' \neq x$.

When describing probabilistic algorithms, we write $x \leftarrow A(y)$ to denote that x is assigned the result of evaluating the probabilistic algorithm A on input y . We write $x \stackrel{R}{\leftarrow} M$ to denote that x is assigned a uniformly chosen element from the set M . A set S is called *efficiently decidable* if there is a deterministic polynomial-time algorithm A such that $A(x) = 1$ iff $x \in S$ (we assume some natural encoding of the elements of S as bitstrings). A set S of lists is *prefix-closed* if for any x, y where $x \in S$ and y is a prefix of x , we have that $y \in S$.

When writing rules of the form $\frac{A \quad B}{C}$, we mean $A \wedge B \Rightarrow C$. The form $\frac{A \quad B}{C}$ is more readable for complex rules.

By 1^n and 0^n with $n \in \mathbb{N}$ we denote bitstrings $11 \dots 1$ and $00 \dots 0$ of length n .

A *substitution* is a partial function from variables (or, in some cases, other symbols) to terms. Given a term T , we write $T\sigma$ for the result of applying the substitution σ to T . That is, $T\sigma$ is the result of replacing every variable $x \in \text{dom } \sigma$ occurring in T by $\sigma(x)$. Example: If $\sigma(x) = f(y)$, $\sigma(y) = z$, and σ undefined otherwise, then $g(x, y, w)\sigma = g(f(y), z, w)$. To specify substitutions with finite domain, we use the following notation: a substitution mapping x_1 to t_1, \dots, x_n to t_n is written $\{t_1/x_1, \dots, t_n/x_n\}$. Example: The substitution σ from the example before is written $\{f(y)/x, z/y\}$.

A *context* is a term with a special symbol \square , the so-called *hole*. Given a context C and a term t , we write $C[t]$ for the result of replacing every occurrence of \square in C by t .

2 Example: Protocol analysis

Consider the following protocol:

- Setting: We have parties Alice and Bob. enc is a symmetric encryption scheme (we write $enc(K, m)$ for an encryption of the plaintext m under key K). We assume that a key K_1 is pre-shared between Alice and Bob.
- Bob chooses keys K_2, K_3 and sends $K_3, enc(K_1, K_2)$ to Alice.
- Alice chooses a nonce¹ N and sends $enc(K_3, enc(K_2, N))$ to Bob.

We assume that the adversary is passive, i.e., he merely observes the messages sent by Alice and Bob. The goal of the adversary is to guess N .

2.1 Cryptographer's analysis

Definition 1 (IND-OT-CPA) *A symmetric encryption scheme $(enc, dec, keygen)$ is said to be IND-OT-CPA secure (indistinguishable under one-time chosen-plaintext attacks) iff for any pair of probabilistic polynomial-time² algorithms Adv_1, Adv_2 with the property that Adv_1 always outputs triples $(m_0, m_1, state)$ with $|m_0| = |m_1|$, there is a negligible function μ such that for all $k \in \mathbb{N}$,*

$$\left| \Pr[out = 1 : Exp_0] - \Pr[out = 1 : Exp_1] \right| \leq \mu(k)$$

where

$$\begin{aligned} Exp_b := & (K \leftarrow keygen(1^k), (m_0, m_1, state) \leftarrow Adv_1(1^k), \\ & c \leftarrow enc(1^k, K, m_b), out \leftarrow Adv_2(1^k, state, c)) \end{aligned}$$

Definition 2 (Security) *The protocol described in this section is secure iff for any probabilistic polynomial-time algorithm Adv there is a negligible function μ such that*

$$\begin{aligned} \Pr[N' = N : N \xleftarrow{R} \{0, 1\}^k, K_1 \leftarrow keygen(1^k), K_2 \leftarrow keygen(1^k), K_3 \leftarrow keygen(1^k), \\ c_1 \leftarrow enc(1^k, K_1, K_2), c_2 \leftarrow enc(1^k, K_3, enc(1^k, K_2, N)), \\ N' \leftarrow Adv(1^k, K_3, c_1, c_2)] \leq \mu(k) \end{aligned}$$

Theorem 1 *If $(enc, dec, keygen)$ is an IND-OT-CPA secure symmetric encryption scheme and $enc, keygen$ are probabilistic polynomial-time algorithms, then we have security in the sense of Definition 2.*

¹A nonce is – roughly speaking – a random value that is long enough to be unguessable.

²Unless specified otherwise, we call an algorithm polynomial-time if there is a polynomial p such that the probability that the algorithm runs more than $p(k)$ steps is 0. Here k is the length of the first argument given to the algorithm.

2.2 Analysis in the Dolev-Yao model

Definition 3 (Messages) A message is a term M of the following grammar:

$$M ::= N \mid enc(M, M) \mid K_1 \mid K_2 \mid K_3.$$

where N, K_1, K_2, K_3 are symbols and enc is a binary function symbol (a constructor).

Definition 4 (Deduction relation) We define the deduction relation \vdash . Given a set S of messages and a message m , $S \vdash m$ intuitively means that m is deducible from the messages in S . Formally, the deduction relation \vdash is the smallest relation that satisfies the following rules:

$$\frac{m \in S}{S \vdash m} \quad \frac{S \vdash m_1 \quad m_1 \in \{K_1, K_2, K_3\} \quad S \vdash m_2}{S \vdash enc(m_1, m_2)} \quad \frac{S \vdash m_1 \quad S \vdash enc(m_1, m_2)}{S \vdash m_2}$$

Definition 5 (Security) The protocol described in this section is secure iff $S \not\vdash N$ where $S := \{K_3, enc(K_1, K_2), enc(K_3, enc(K_2, N))\}$.

Theorem 2 We have security in the sense of Definition 5.

Lecture on
2009-10-20

3 The Dolev-Yao approach: counterexamples

In this section, we give a few examples for Dolev-Yao models where the symbolic analysis claims that a given protocol is secure, while it is easy to break given the actual cryptographic implementation.

First, we give several models for cryptography using groups:

Definition 6 (Symbolic model 1) The set of messages M is defined by

$$M ::= N \mid N^A \mid enc(M, M) \mid g(M) \mid f(M, M)$$

where N and N^A are two different countably infinite sets of symbols (protocol nonces and adversary nonces, respectively).

The intuitive computational meaning of g is that $g(x)$ represents the group element g^x where g is a generator of a group G . The intuitive computational meaning of f is that $f(x, y)$ represents the group element x^y .

Let \approx be the smallest equivalence relation on messages such that the following rules hold:

$$f(g(M_1), M_2) \approx f(g(M_2), M_1) \quad \frac{M_1 \approx M_2 \quad C \text{ is a context}}{C[M_1] \approx C[M_2]}$$

where M_1, M_2 are messages.

Let \vdash be the smallest relation on sets of messages and messages such that the following rules hold:

$$\frac{M \in S}{S \vdash M} \quad \frac{S \vdash M_1, M_2}{S \vdash \text{enc}(M_1, M_2), g(M_1), f(M_1, M_2)} \quad \frac{N \in N_A}{S \vdash N}$$

$$\frac{S \vdash M_1, \text{enc}(M_1, M_2)}{S \vdash M_2} \quad \frac{S \vdash M \quad M \approx M'}{S \vdash M'}$$

(Note: the last rule has the effect that any two equivalent terms are treated the same with respect to the deduction relation.)

(This symbolic model has been suggested in [AF01].)

Definition 7 (Example protocol 1) Bob picks nonces X, Y, N and sends $X, f(g(X), Y), \text{enc}(g(Y), N)$ to Alice.

Lemma 1 The protocol from Definition 7 keeps N secret with respect to the symbolic model from Definition 6. That is, with $S := \{X, f(g(X), Y), \text{enc}(g(Y), N)\}$, we have $S \not\vdash N$.

The protocol from Definition 7 can be broken in the computational model if the underlying group G has known order: Given the messages X and $t_1 := f(g(X), Y) = g^{XY}$, the adversary can compute $g(Y) = g^Y = t_1^{X^{-1} \bmod \text{ord } G}$. Then $g(Y)$ can be used to decrypt $\text{enc}(g(Y), N)$. However, this only works if $\text{ord } G$ is known. The next example considers the case where $\text{ord } G$ is unknown (such as in, e.g., the RSA-group).

Definition 8 (Example protocol 1b) Alice picks a nonce X and sends X to Bob. Bob picks nonces Y, N and replies with $f(g(Y), X), \text{enc}(g(Y), N)$.

Lemma 2 The protocol from Definition 8 keeps N secret with respect to the symbolic model from Definition 6 even when Alice is corrupted. That is, for any X^* with $\emptyset \vdash X^*$ we have $\{f(g(Y), X^*), \text{enc}(g(Y), N)\} \not\vdash N$. (Note: X^* does not have to be a nonce!)

Computationally, the protocol can be broken even when the order of the group G is not known: The adversary sets $X^* := 1$ and sends X^* to Bob. Then Bob answers with $f(g(Y), 1) = g^{Y \cdot 1} = g(Y)$ and $\text{enc}(g(Y), N)$ from which the adversary can decrypt N .

To avoid such examples, we extend the symbolic model to include symbols *zero* and *one*.

Definition 9 (Symbolic model 2) The set of messages M is defined by

$$M ::= N | N^A | \text{enc}(M, M) | g(M) | f(M, M) | \text{zero} | \text{one}$$

The equivalence relation is defined as in Definition 6, except that we add the following rules:

$$f(x, \text{one}) = x \quad f(x, \text{zero}) = \text{one}$$

The deduction relation is defined as in Definition 6, except that we add the following rule:

$$S \vdash \text{zero}, \text{one}$$

Definition 10 (Example protocol 2) Alice picks nonces X, Y and sends them to Bob. Bob checks that $X, Y \not\approx \text{zero}$ and $X, Y \not\approx \text{one}$ and $X \not\approx Y$. Otherwise, Bob aborts. Bob picks nonces W, Z, N and replies with

$$t_1 := f(g(W), X), \quad t_2 := f(g(W), Y), \quad t_3 := \text{enc}(f(g(W), Z), N), \quad Z$$

Lemma 3 The protocol from Definition 10 keeps N secret with respect to the symbolic model from Definition 9 even when Alice is corrupted. That is, for any X^*, Y^* with $\emptyset \vdash X^*, Y^*$ and $X^*, Y^* \not\approx \text{zero}$ and $X^*, Y^* \not\approx \text{one}$ and $X^* \not\approx Y^*$ we have

$$\{f(g(W), X^*), f(g(W), Y^*), \text{enc}(f(g(W), Z), N), Z\} \not\vdash N$$

(Note: X^*, Y^* do not have to be nonces!)

Computationally, the protocol can be broken even when the order of the group G is not known: The adversary chooses X^* at random and sets $Y^* := X^* + 1$. He sends X^*, Y^* to Bob. Then Bob answers with $t_1 := f(g(W), X^*) = g^{WX^*}$, $t_2 := f(g(W), Y^*) = g^{WY^*} = g^{WX^*} g^W = t_1 g^W$, $c := \text{enc}(f(g(W), Z), N)$, Z . Then the adversary can compute $f(g(W), Z)$ as $(t_2/t_1)^Z$ and decrypt N .

Side note: It should be mentioned that in the case of *passive* adversaries, computational soundness results for certain classes of protocols involving group arithmetic are known. See, e.g., [BLMW07, BCK05].

Our last example is concerned with the XOR operation instead of group theory.

Definition 11 (Symbolic model for XOR) The set of messages M is defined by

$$M ::= N | N^A | \text{xor}(M, M) | \text{zero}$$

where N and N^A are two different countably infinite sets of symbols (protocol nonces and adversary nonces, respectively).

Let \approx be the smallest equivalence relation on messages such that the following rules hold:

$$\begin{aligned} \text{xor}(M_1, M_2) &\approx \text{xor}(M_2, M_1) & \text{xor}(M_1, \text{xor}(M_2, M_3)) &\approx \text{xor}(\text{xor}(M_1, M_2), M_3) \\ \text{xor}(M_1, M_1) &\approx \text{zero} & \text{xor}(M_1, \text{zero}) &\approx M_1 & \frac{M_1 \approx M_2 \quad C \text{ is a context}}{C[M_1] \approx C[M_2]} \end{aligned}$$

where M_1, M_2, M_3 are messages.

Let \vdash be the smallest relation on sets of messages and messages such that the following rules hold:

$$\frac{M \in S}{S \vdash M} \quad \frac{S \vdash M_1, M_2}{S \vdash \text{xor}(M_1, M_2)} \quad \frac{N \in N_A}{S \vdash N} \quad \frac{S \vdash M \quad M \approx M'}{S \vdash M'}$$

Note: we have not included encryption in this model for simplicity. But encryption could be easily added analogously as in the previous symbolic models.

Definition 12 (Example protocol for XOR) *First Bob picks a nonce N_{secret} . When Bob gets the i -th query getnonce, he picks a nonce N_i and sends N_i to Alice. When Bob gets the first query getmaster, he picks a nonce M and sends M to Alice. When Bob gets a list of nonces (M_1, \dots, M_m) , he checks whether $\{M_1, \dots, M_m\} \subseteq \{N_1, \dots, N_n\}$ and $xor(M_1, xor(M_2, xor(M_3, \dots xor(M_{m-1}, M_m)))) = M$. (Here n is the number of getnonce-queries that Bob got.) If this check succeeds, Bob sends N_{secret} to Alice.*

Note: When analysing this protocol with respect to some symbolic model (e.g., Definition 11), all equality tests $t_1 = t_2$ performed by Bob are interpreted as tests whether $t_1 \approx t_2$ because the intuition behind \approx is that two terms are equivalent iff they are “the same”.

Definition 13 (Reasonable theories for XOR) *Assume a set of terms containing a binary constructor xor and infinitely many nonces. We say an equivalence relation \approx reasonable iff*

- *For any permutation σ on the set of all nonces and any two messages M_1, M_2 , we have $M_1 \approx M_2$ iff $M_1\sigma \approx M_2\sigma$.*
- *There are infinitely many nonces that are pairwise different with respect to \approx .*

We call \vdash reasonable iff $S \not\vdash N_{secret}$ for all sets S that contain only messages not containing N_{secret} .

The symbolic model from Definition 11 fulfils this definition.

Lemma 4 (informal) *With respect to reasonable \approx and \vdash , when Alice is corrupted, the adversary cannot learn N_{secret} . That is, for any sequence of messages the adversary can send such that any message sent by the adversary can be deduced from the messages he received before, we have that N_{secret} cannot be deduced from the messages the adversary receives.*

Computationally, however, the protocol can be broken: Let k be the maximum length of the nonces. Then the nonces span an at most k -dimensional vector space V over $\text{GF}(2)$. Let $n := 2k$. The adversary requests nonces N_1, \dots, N_n . With probability $\geq \frac{1}{2}$, the nonces N_1, \dots, N_n span the whole vector space V . Thus $M \in V$ is a linear combination of N_1, \dots, N_n . Thus there is a subset $\{M_1, \dots, M_m\}$ of $\{N_1, \dots, N_n\}$ such that $M_1 \oplus \dots \oplus M_m = M$. By sending (M_1, \dots, M_m) , the adversary gets N_{secret} from Bob.

Side note: It should be mentioned that in the case of *passive* adversaries, computational soundness results for certain classes of protocols involving XOR are known. See, e.g., [BCK05].

Summing up. The aim of the examples presented in this section is not to indicate that the Dolev-Yao-approach is useless in principle. They should, however, make you aware of the pitfalls in using the Dolev-Yao-approach for protocol analysis. At the very least, when designing a symbolic model one should very carefully check whether the model makes sense, it is not enough to write down the rules that immediately spring to mind. A better approach is, of course, to prove that a given symbolic model is reasonable; to this end, we will investigate so-called computational soundness theorems later in this lecture.

4 Inference Rules and Induction

We call a function F from sets to sets *monotone* iff for all sets X, X' with $X \subseteq X'$ we have $F(X) \subseteq F(X')$. We call F *bounded* if there exists a set U (the universe) such that for any set X , $F(X) \subseteq U$.

Definition 14 Let F be a function from sets to sets. The least fixed point $\text{lfp}(F)$ of F is the smallest set X such that $F(X) = X$. More precisely: $F(\text{lfp}(F)) = \text{lfp}(F)$ and for any X with $F(X) = X$ we have $\text{lfp}(F) \subseteq X$.

Theorem 3 (Knaster-Tarski) If F is monotone and bounded, then F has a least fixed point. In fact, $\text{lfp}(F) = \bigcap_{X \text{ with } F(X) \subseteq X} X$.

Theorem 4 (Induction principle) Let F be monotone and bounded. Let P be a set. Assume that $F(P) \subseteq P$. Then $\text{lfp}(F) \subseteq P$.

An (inference) *rule* is a formula of the form $\frac{\text{premise}}{\text{expression} \in S}$. Here *premise* and *expression* may contain free variables, *premise* may contain occurrences of S , but *expression* may not contain S .

Given a set of such rules $\frac{\text{premise}_i}{\text{expression}_i \in S}$, $i = 1, \dots, n$ we define a function F as follows:

$$F(X) := X \cup \bigcup_{i=1}^n \{\text{expression}_i : \text{premise}_i\{X/S\} \text{ holds}\}.$$

Here $\text{premise}_i\{X/S\}$ denotes the result of replacing all free occurrences of S in premise_i by X (we assume that the variable X does not occur in the rules).

We call a rule *bounded* if there is a set U such that for any set X , $\{\text{expression} : \text{premise}\{X/S\} \text{ holds}\} \subseteq U$. We call a rule *monotone* if for $X \subseteq X'$, we have that $\text{premise}\{X/S\} \implies \text{premise}\{X'/S\}$.

When we say, “let S be the smallest set satisfying certain rules”, then we mean $S := \text{lfp}(F)$ where F is the function F corresponding to the rules. If all rules are bounded and monotone, so is F , and hence S exists.

Theorem 5 (Rule induction) Let bounded and monotone rules $\frac{\text{premise}_i}{\text{expression}_i \in S}$, $i = 1, \dots, n$ be given. Let S be the smallest set satisfying these rules. Let P be a set. Assume that the following holds for each $i = 1, \dots, n$:

$$\text{premise}_i\{P/S\} \implies \text{expression}_i \in P$$

Then $S \subseteq P$.

In some cases, the set being defined might be a relation, e.g., \vdash . In this case, one considers a relation as the set of pairs that satisfy the relation. (E.g., $\vdash = \{(x, y) : x \vdash y\}$.) If no premise is given, then *true* is assumed as premise (e.g., $0 \in S$ means $\frac{\text{true}}{0 \in S}$). If the conclusion of the rule lists several elements in the set, this is just a shorthand for several rules, e.g., $\frac{\dots}{0 \in S \quad 1 \in S}$ is short for the two rules $\frac{\dots}{0 \in S}$ and $\frac{\dots}{1 \in S}$.

See also [Pie02, Section 21.1] for more information on induction.

5 Symbolic models

Definition 15 (Constructors, destructors, nonces, and messages types) A constructor C is a symbol with a (possibly zero) arity. A nonce N is a symbol with zero arity. We write $C/n \in \mathbf{C}$ to denote that \mathbf{C} contains a constructor C with arity n . A message type \mathbf{T} over \mathbf{C} and \mathbf{N} is a set of terms over constructors \mathbf{C} and nonces \mathbf{N} . A destructor D of arity n , written D/n , over a message type \mathbf{T} is a partial function $\mathbf{T}^n \rightarrow \mathbf{T}$. If D is undefined on t_1, \dots, t_n , we write $D(t_1, \dots, t_n) = \perp$.

Definition 16 (Deduction relation) A deduction relation \vdash over a message type \mathbf{T} is a relation between $2^{\mathbf{T}}$ and \mathbf{T} .

Definition 17 A symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ consists of a set of constructors \mathbf{C} , a set of nonces \mathbf{N} , a message type \mathbf{T} over \mathbf{C} and \mathbf{N} with $\mathbf{N} \subseteq \mathbf{T}$, a set of destructors \mathbf{D} over \mathbf{T} , and a deduction relation \vdash over \mathbf{T} .

Definition 18 (Standard nonces and standard deduction relation) The standard nonces are $\mathbf{N} := \mathbf{N}^A \cup \mathbf{N}^P$ where \mathbf{N}^A and \mathbf{N}^P are disjoint countably infinite sets. (\mathbf{N}^A is the set of adversary nonces and \mathbf{N}^P the set of protocol nonces.)

The standard deduction relation is defined to be the smallest relation satisfying the rules given in Figure 1.

Further reading: [BHU09, Section 2].

Lecture on
2009-11-03

$$\begin{array}{c}
\frac{m \in S}{S \vdash m} \quad \frac{N \in \mathbf{N}^A}{S \vdash N} \quad \frac{C/n \in \mathbf{C} \quad S \vdash m_1, \dots, m_n \quad C(m_1, \dots, m_n) \in \mathbf{T}}{S \vdash C(m_1, \dots, m_n)} \\
\\
\frac{D/n \in \mathbf{D} \quad S \vdash m_1, \dots, m_n \quad D(m_1, \dots, m_n) \neq \perp}{S \vdash D(m_1, \dots, m_n)}
\end{array}$$

Figure 1: Deduction rules for the standard deduction relation. In the last rule, $D(m_1, \dots, m_n)$ denotes the term resulting from applying the function D to m_1, \dots, m_n .

6 Computational soundness: The passive case

Definition 19 (Passive secrecy property) *A passive secrecy property (with respect to some symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$) is a pair $\wp := (N_{sec}, (m_1, \dots, m_n))$ with $N_{sec} \in \mathbf{N}$ and $m_1, \dots, m_n \in \mathbf{T}$.*

We say \wp holds symbolically iff $\{m_1, \dots, m_n\} \not\vdash N_{sec}$.

Note: In this definition, we consider a protocol secure if the adversary cannot guess the nonce N_{sec} . Such a secrecy property (also in the case of active adversaries) is often called *weak secrecy*. In contrast, *strong secrecy* would mean that the adversary cannot recognise N_{sec} , even when he sees it.

Definition 20 (Computational implementation) *Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be given. A computational implementation of \mathbf{M} is a family of functions $A = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}}$ such that A_F for $F/n \in \mathbf{C} \cup \mathbf{D}$ is a partial deterministic function $\mathbb{N} \times (\{0, 1\}^*)^n \rightarrow \{0, 1\}^*$, and A_N for $N \in \mathbf{N}$ is a total probabilistic function with domain \mathbb{N} and range $\{0, 1\}^*$ (i.e., it specifies a probability distribution on bitstrings that depends on its argument). The first argument of A_F and A_N represents the security parameter.*

*All functions A_F have to be computable in deterministic polynomial-time, and all A_N have to be computable in probabilistic polynomial-time.*³

In the following, we always omit the first argument of A_F , $F \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}$ (the security parameter). Keep in mind that A_F always implicitly gets the security parameter k as its first argument.

Definition 21 (Computational execution of a p.s.p.) *Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ and a passive secrecy property $\wp = (N_{sec}, (m_1, \dots, m_n))$ be given. Fix an algorithm Adv . Fix a security parameter k . Consider the following algorithm:*

³More precisely, there has to exist a single uniform probabilistic polynomial-time algorithm A that, given the name of $C \in \mathbf{C}$, $D \in \mathbf{D}$, or $N \in \mathbf{N}$, together with an integer k and the inputs m_1, \dots, m_n , computes the output of A_C , A_D , and A_N or determines that the output is undefined. This algorithm must run in polynomial-time in $k + |m_1| + \dots + |m_n|$ and may not use random coins when computing A_C and A_D .

- For each nonce N occurring in \wp (i.e., N_{sec} or any subterm $N \in \mathbf{N}$ of some m_i), choose $r_N \leftarrow A_N$.
- For $i = 1, \dots, n$, compute $b_i := \beta(m_i)$. Here β is the recursive function that is defined as follows:
 - $\beta(N) := r_N$ for $N \in \mathbf{N}$.
 - $\beta(F(m_1, \dots, m_t)) := A_F(\beta(m_1), \dots, \beta(m_t))$ for $F/t \in \mathbf{C}$.
- Invoke $n^* \leftarrow \text{Adv}(1^k, b_1, \dots, b_n)$.
- If $n^* = r_{N_{sec}}$, we say the adversary wins.

We define $\text{Succ}(\wp, \text{Adv}, k)$ as the probability that the adversary wins in the above algorithm.

Definition 22 (P.s.p. holds computationally) Let a passive secrecy property \wp be given. We say \wp holds computationally if for any polynomial-time algorithm Adv we have that $\text{Succ}(\wp, \text{Adv}, k)$ is negligible as a function of k .

Definition 23 (Symbolic model for symmetric encryption) Let $\mathbf{C} ::= \{\text{enc}/3, \text{pair}/2\}$. Let \mathbf{N} be standard nonces. Let \mathbf{T} be defined by $\mathbf{T} ::= \mathbf{N} | \text{enc}(\mathbf{N}, \mathbf{T}, \mathbf{N}) | \text{pair}(\mathbf{T}, \mathbf{T})$. Let $\mathbf{D} := \{\text{dec}/2, \text{fst}/1, \text{snd}/1\}$ with

$$\begin{aligned} \text{dec}(x, \text{enc}(x, y, z)) &= y \\ \text{fst}(\text{pair}(x, y)) &= x \\ \text{snd}(\text{pair}(x, y)) &= y. \end{aligned}$$

(If none of these rules apply, the destructor application evaluates to \perp .) Let \vdash be the standard deduction relation.

The symbolic model for symmetric encryption is $\mathbf{M}_{enc} := (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$.

Definition 24 (Key-cycle-freeness) Let \wp be a passive secrecy property with respect to \mathbf{M}_{enc} . We call a nonce N hidden if N occurs in \wp and $m_1, \dots, m_n \not\vdash N$.

We say a nonce N' occurs as a non-key in a term $M \in \mathbf{T}$ if M is of the following grammar: $M ::= N' | \text{pair}(M, \mathbf{T}) | \text{pair}(\mathbf{T}, M) | \text{enc}(\mathbf{N}, M, \mathbf{T})$. (That is, N' occurs in M somewhere else than first and third argument of enc .)

We say a nonce N' is encrypted with a nonce N if there is a subterm $\text{enc}(N, M, R)$ of some m_i such that N' occurs as a non-key in M .

We say \wp is key-cycle-free if the hidden nonces can be ordered as $N_1 < N_2 < \dots < N_t$ such that the following holds:

For any two hidden nonces N_i, N_j , if N_i is encrypted with N_j in \wp , then $N_i > N_j$.

Definition 25 (Length regularity) A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ is length-regular, if:

- For any constructor $C/n \in \mathbf{C}$, any $k \in \mathbb{N}$, any bitstrings $m_1, \dots, m_n, m'_1, \dots, m'_n \in \{0, 1\}^*$ with $|m_1| = |m'_1|, \dots, |m_n| = |m'_n|$ and $A_C(1^k, m_1, \dots, m_n) \neq \perp \neq A_C(1^k, m'_1, \dots, m'_n)$, we have that $|A_C(1^k, m_1, \dots, m_n)| = |A_C(1^k, m'_1, \dots, m'_n)|$.
- For any nonce $N \in \mathbf{N}$, any $k \in \mathbb{N}$, there is an $l \in \mathbb{N}$ such that $\Pr[|x| = l : x \leftarrow A_N(1^k)] = 1$.

Definition 26 (Failure-freeness) A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ is failure-free, if for any $M \in \mathbf{T}$ containing nonces $N_1, \dots, N_n \in \mathbf{N}$ we have that $\Pr[\beta(M) = \perp]$ is negligible as a function of the security parameter k . Here β is defined as in Definition 21.

Definition 27 (Unpredictable nonces) A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ has unpredictable nonces if for all $N \in \mathbf{N}$, we have that $\max_x \Pr[A_N(1^k) = x]$ is negligible as a function of k .⁴

Definition 28 (Fresh randomness) A passive secrecy property $\wp = (N_{secret}, (m_1, \dots, m_n))$ with respect to \mathbf{M}_{enc} uses fresh randomness if the following holds for all nonces $R \in \mathbf{N}$: If m_i has a subterm $enc(M_1, M_2, R)$ for some i, M_1, M_2 , then R occurs exactly once as a subterm of (m_1, \dots, m_n) .

Note: This definition could be weakened to permit that the same term $enc(N, M, R)$ with the same key, plaintext, and randomness occurs more than once. For simplicity, we omit this generalisation.

Definition 29 (IND-CPA) An encryption scheme (K, E, D) consisting of probabilistic polynomial-time algorithms K (key-generation), E (encryption), and D (decryption) is IND-CPA secure if for any polynomial-time oracle machine Adv , there is a negligible function μ such that for all k we have $|\Pr_0(k) - \Pr_1(k)| \leq \mu(k)$. Here

$$\Pr_i(k) := \Pr[b = 1 : \text{key} \leftarrow K(1^k), b \leftarrow \text{Adv}^{\mathcal{E}_i(\text{key}, \cdot)}(1^k)].$$

and $\mathcal{E}_0(\text{key}, \cdot)$ is an oracle that returns $E(1^k, \text{key}, m)$ when queried with a message m , and $\mathcal{E}_1(\text{key}, \cdot)$ is an oracle that returns $E(1^k, \text{key}, 0^{|m|})$ when queried with a message m .

(The relevant difference to IND-OT-CPA as defined in Definition 1 is that in the present definition, the adversary may request the encryptions of many messages, while in the IND-OT-CPA definition, he may request only one.)

Definition 30 (IND-CPA (comp. implementation)) A computational implementation A of \mathbf{M}_{enc} is IND-CPA secure if (K, E, D) is IND-CPA secure where

- All A_N for $N \in \mathbf{N}$ are identical (i.e., A_N and A_M are the same algorithm for $N, M \in \mathbf{N}$).

⁴In other words, the min-entropy $H_\infty(A_N(1^k))$ is superlogarithmic.

- $K(1^k) := A_N(1^k)$ for some $N \in \mathbf{N}$.
- $E(1^k, key, m) := A_{enc}(1^k, key, m, A_N(1^k))$ for some $N \in \mathbf{N}$.

Theorem 6 (Passive computational soundness of symmetric encryption) *Let $\wp = (N_{secret}, (m_1, \dots, m_n))$ be a passive secrecy property with respect to \mathbf{M}_{enc} . Let A be a computational implementation of \mathbf{M}_{enc} . Assume the following facts:*

- \wp is key-cycle-free (Definition 24).
- A is length-regular (Definition 25).
- A is failure-free (Definition 26).
- A has unpredictable nonces (Definition 27).
- \wp has no subterm of the form $enc(N_{secret}, M_1, M_2)$ or $enc(M_1, M_2, N_{secret})$ for some M_1, M_2 .
- \wp uses fresh randomness (Definition 28).
- A is IND-CPA secure (Definition 30).

Then, if \wp holds symbolically, \wp holds computationally.

Actually, the fact that A has unpredictable nonces follows from the fact that A is IND-CPA secure. (If we could guess the key, we could break the IND-CPA security.) We have included the condition anyway to keep the proof simpler, and to stress the fact that unpredictability is a property used independently of the IND-CPA security. E.g., if we would not have any encryptions, we would not need IND-CPA security, but we would still need unpredictable nonces.

A result similar to Theorem 6 is developed in [AR02]. However, the details are different. In particular, they focus on indistinguishability properties, not on weak secrecy properties as we do.

**Lecture on
2009-11-24**

7 Computational soundness: The active case

The modelling presented in this section closely follows that of [BHU09].

7.1 Protocol model

First, we define a model which allows to express arbitrary protocols. We call these protocols *CoSP protocols* (CoSP stands for *Computational Soundness Proofs*).

We define a CoSP protocol as a tree with a distinguished root and with labels on both nodes and edges. Intuitively, the nodes correspond to different protocol actions: *Computation nodes* produce terms (using a constructor or destructor); *input and output nodes* correspond to receive and send operations; *nondeterministic nodes* encode nondeterministic choices in the protocol; *control nodes* allow an outside entity (i.e., an adversary) to influence the control flow of the protocol.

The edge labels intuitively allow for distinguishing branches in the protocol execution, e.g., destructor nodes have two outgoing edges labelled with *yes* and *no*, corresponding

to the two cases that the destructor is defined on the input term or not; hence we can, e.g., speak about the *yes*-successor of a destructor node.

Definition 31 (CoSP protocol) *A CoSP protocol Π is a tree with a distinguished root and labels on both edges and nodes. Each node has a unique identifier N and one of the following types:*

- *Computation nodes are annotated with a constructor, nonce, or destructor F/n together with the identifiers of n (not necessarily distinct) nodes. Computation nodes have exactly two successors; the corresponding edges are labeled with *yes* and *no*, respectively.*
- *Output nodes are annotated with the identifier of one node. An output node has exactly one successor.*
- *Input nodes have no further annotation. An input node has exactly one successor.*
- *Control nodes are annotated with a bitstring l . A control node has at least one and up to countably many successors annotated with distinct bitstrings $l' \in \{0, 1\}^*$. (We call l the *out-metadata* and l' the *in-metadata*.)*
- *Nondeterministic nodes have no further annotation. Nondeterministic nodes have at least one and at most finitely many successors; the corresponding edges are labeled with distinct bitstrings.*

*We assume that the annotations are part of the node identifier N . If a node N contains an identifier N' in its annotation, then N' has to be on the path from the root to N (including the root, excluding N), and N' must be a computation node or input node. In case N' is a computation node, the path from N' to N has to additionally go through the outgoing edge of N' with label *yes*.*

7.2 Symbolic execution

The symbolic execution of a CoSP protocol models what can happen in an execution of the protocol in the symbolic model when running with an active adversary.

The symbolic execution of a CoSP protocol for a given symbolic model consists of a sequence of triples (S, ν, f) where S represents the knowledge of the adversary, ν represents the current node identifier in the protocol, and f represents a partial function mapping already processed node identifiers to messages.

In the following, for a constructor, destructor, or nonce $F/n \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}$, we define $eval_F(t_1, \dots, t_n)$ as follows:

$$\begin{array}{ll}
 eval_F() := F & \text{if } F \text{ is a nonce} \\
 eval_F(t_1, \dots, t_n) := F(t_1, \dots, t_n) & \text{if } F \text{ is a destructor and} \\
 & F(t_1, \dots, t_n) \neq \perp \\
 eval_F(t_1, \dots, t_n) := F(t_1, \dots, t_n) & \text{if } F \text{ is a constructor and} \\
 & F(t_1, \dots, t_n) \in \mathbf{T} \\
 eval_F(t_1, \dots, t_n) := \perp & \text{otherwise}
 \end{array}$$

Definition 32 (Symbolic execution) Let a symbolic model $(\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ and a CoSP protocol Π be given. A full trace is a (finite) list of tuples (S_i, ν_i, f_i) such that the following conditions hold:

- Correct start: $S_1 = \emptyset$, ν_1 is the root of Π , f_1 is a totally undefined partial function mapping node identifiers to terms.
- Valid transition: For every two consecutive tuples (S, ν, f) and (S', ν', f') in the list, let $\tilde{\nu}_1, \dots, \tilde{\nu}_n$ be the node identifiers in the annotation of ν and define $\tilde{t}_j := f(\tilde{\nu}_j)$. We have:
 - If ν is a computation node with constructor, destructor or nonce F , then $S' = S$. If $m := \text{eval}_F(\tilde{t}_1, \dots, \tilde{t}_n) \neq \perp$, ν' is the yes-successor of ν in Π , and $f' = f(\nu := m)$. If $m = \perp$, then ν' is the no-successor of ν and $f' = f$.
 - If ν is an input node, then $S' = S$ and ν' is the successor of ν in Π and there exists an m with $S \vdash m$ and $f' = f(\nu := m)$.
 - If ν is an output node, then $S' = S \cup \{\tilde{t}_1\}$, ν' is the successor of ν in Π and $f' = f$.
 - If ν is a control or a nondeterministic node, then ν' is a successor of ν and $f' = f$ and $S' = S$.

A list of node identifiers (ν_i) is a node trace if there is a full trace with these node identifiers.

Lecture on
2009-12-01

Definition 33 (Trace property) A trace property \wp is an efficiently decidable and prefix-closed set of (finite) lists of node identifiers.

Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be a symbolic model and Π a CoSP protocol. Then Π symbolically satisfies a trace property \wp in \mathbf{M} iff every node trace of Π is contained in \wp .

7.3 Computational execution

Definition 34 (Probabilistic CoSP protocol) A probabilistic CoSP protocol Π is a CoSP protocol, where each nondeterministic node is additionally annotated with a probability distribution over the labels of the outgoing edges.

Definition 35 (Computational execution) Let a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$, a computational implementation A of \mathbf{M} , and a probabilistic CoSP protocol Π be given. Let a probabilistic polynomial-time interactive machine Adv (the adversary) be given (polynomial-time in the sense that the number of steps in all activations are bounded in the length of the first input of Adv), and let p be a polynomial. We define a probability distribution $\text{Nodes}_{\mathbf{M}, A, \Pi, \text{Adv}}^p(k)$ (the computational node trace) over (finite) lists of node identifiers (ν_i) according to the following probabilistic algorithm (both the algorithm and Adv are run on input k):

- Initial state: $\nu_1 := \nu$ is the root of Π . Let f be an initially empty partial function from node identifiers to bitstrings, and let r be an initially empty partial function from \mathbf{N} to bitstrings.
- For $i = 2, 3, \dots$ do the following:
 - Let $\tilde{\nu}_1, \dots, \tilde{\nu}_n$ be the node identifiers in the annotation of ν . $\tilde{m}_j := f(\tilde{\nu}_j)$.

- Proceed depending on the type of node ν :
 - * If ν is a computation node with nonce $N \in \mathbf{N}$: Let $m' := r(N)$ if $r(N) \neq \perp$ and sample m' according to $A_N(k)$ otherwise. Let ν' be the yes-successor of ν , $f' := f(\nu := m')$, and $r' := r(N := m')$.
 - * If ν is a computation node with constructor or destructor F , then $m' := A_F(k, \tilde{m}_1, \dots, \tilde{m}_n)$. If $m' \neq \perp$, then ν' is the yes-successor of ν , if $m' = \perp$, then ν' is the no-successor of ν . Let $f' := f(\nu := m')$. Let $r' := r$.
 - * If ν is an input node, ask for a bitstring m from Adv. Abort the loop if Adv halts. Let ν' be the successor of ν . Let $f' := f(\nu := m)$ and $r := \nu'$.
 - * If ν is an output node, send \tilde{m}_1 to Adv. Abort the loop if Adv halts. Let ν' be the successor of ν . Let $f' := f$ and $r' := r$.
 - * If ν is a control node, annotated with out-metadata l , send l to Adv. Abort the loop if Adv halts. Upon receiving an answer l' , let ν' be the successor of ν along the edge labeled l' (or the lexicographically smallest edge if there is no edge with label l'). Let $f' := f$ and $r' := r$.
 - * If ν is a nondeterministic node, let \mathcal{D} be the probability distribution in the annotation of ν . Pick ν' according to the distribution \mathcal{D} , and let $\nu := \nu'$. Let $f' := f$ and $r' := r$.
- Let $\nu := \nu'$, $f := f'$ and $r := r'$. Let $\nu_i := \nu$.
- Let len be the number of nodes from the root to ν plus the total length of all bitstrings in the range of f . If $\text{len} > p(k)$, abort.

Definition 36 (Efficient protocol) We call a probabilistic CoSP protocol efficient if:

- There is a polynomial p such that for any node N , the length of the identifier of N is bounded by $p(m)$ where m is the length (including the total length of the edge-labels) of the path from the root to N .
- There is a deterministic polynomial-time algorithm that, given the identifiers of all nodes and the edge labels on the path to a node N , computes the label of N .

Definition 37 (Computationally satisfied trace properties) Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be a symbolic model. Let A be a computational implementation of \mathbf{M} and let Π be a probabilistic CoSP protocol. Then (Π, A) computationally satisfies a trace property \wp in \mathbf{M} iff for all probabilistic polynomial-time interactive machines Adv and all polynomials p , the probability that $\text{Nodes}_{\mathbf{M}, A, \Pi, \text{Adv}}^p(k) \in \wp$ is overwhelming.

Definition 38 (Computational soundness) A computational implementation A of a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ is computationally sound for a class P of CoSP protocols iff for every trace property \wp and for every efficient probabilistic CoSP protocol Π , we have that (Π, A) computationally satisfies \wp if Π_S symbolically satisfies \wp and $\Pi \in P$. Here Π_S denotes the CoSP protocol resulting from removing the distributions at the nondeterministic nodes in the probabilistic CoSP protocol Π .

7.4 Simulators and the hybrid execution

Definition 39 (Simulator) A simulator is an interactive machine Sim that satisfies the following syntactic requirements:

- When activated without input, it replies with a term $m \in \mathbf{T}$. (This corresponds to the situation that the protocol expects some message from the adversary.)
- When activated with some $t \in \mathbf{T}$, it replies with an empty output. (This corresponds to the situation that the protocol sends a message to the adversary.)
- When activated with (info, ν, t) where ν is a node identifier and $t \in \mathbf{T}$, it replies with (proceed) .⁵
- At any point (in particular instead of sending a reply), it may terminate.

Definition 40 (Hybrid execution) Let Π be a probabilistic CoSP protocol for a symbolic model \mathbf{M} , and let Sim be a simulator. We define a probability distribution $H\text{-Trace}_{\mathbf{M}, \Pi, \text{Sim}}(k)$ on (finite) lists of tuples (S_i, ν_i, f_i) called the full hybrid trace according to the following probabilistic algorithm, that runs on input k and interacts with Sim .

- Start: $S_1 := S := \emptyset$, $\nu_1 := \nu$ is the root of Π , and $f_1 := f$ is a totally undefined partial function mapping node identifiers to \mathbf{T} .
- Transition: For $i = 2, 3, \dots$ do the following:
 - Let $\tilde{\nu}_1, \dots, \tilde{\nu}_n$ be the node identifiers in the label of ν . Define $\tilde{t}_1, \dots, \tilde{t}_n$ through $\tilde{t}_j := f(\tilde{\nu}_j)$.
 - Proceed depending on the type of ν :
 - * If ν is a computation node with constructor, destructor, or nonce F , then let $m := \text{eval}_F(\tilde{t}_1, \dots, \tilde{t}_n)$. If $m \neq \perp$, let ν' be the yes-successor of ν and let $f' := f(\nu := m)$. If $m = \perp$, let ν' be the no-successor of ν and let $f' := f$. Set $\nu := \nu'$ and $f := f'$.
 - * If ν is an output node, send \tilde{t}_1 to Sim . Let ν' be the unique successor of ν . Set $\nu := \nu'$.
 - * If ν is an input node, hand control to Sim , and wait to receive $m \in \mathbf{T}$ from Sim . Let $f' := f(\nu := m)$, and let ν' be the unique successor of ν . Set $f := f'$ and $\nu := \nu'$.
 - * If ν is a control node labeled with out-metadata l , send l to Sim , and wait to receive a bitstring l' from Sim . Let ν' be the successor of ν along the edge labeled l' (or the lexicographically smallest edge if there is no edge with label l'). Let $\nu := \nu'$.
 - * If ν is a nondeterministic node, sample ν' according to the probability distribution specified in ν . Let $\nu := \nu'$.
 - Send (info, ν, t) to Sim . When receiving an answer (proceed) from Sim , continue.⁶

⁵This step is necessarily for technical reasons only. See footnote 6.

⁶This step is necessary for technical reasons only. The computational execution may abort at a certain point depending on the polynomial p . The simulator needs to mimic this behaviour, and to know the right time for aborting, he needs to know ν and t .

– If Sim has terminated, stop. Otherwise let $(S_i, \nu_i, f_i) := (S, \nu, f)$.

The probability distribution of the (finite) list ν_1, \dots produced by this algorithm we denote $H\text{-Nodes}_{\mathbf{M}, \Pi, \text{Sim}}(k)$. We call this distribution the hybrid node trace.

Definition 41 (Dolev-Yao style simulator) A simulator Sim is Dolev-Yao style (short: DY) for \mathbf{M} and a probabilistic CoSP protocol Π , if with overwhelming probability the following holds:

In the hybrid execution, for each ℓ , let $m_\ell \in \mathbf{T}$ be the ℓ -th term sent (during processing of one of Π 's input nodes) from Sim to Π in that execution. Let $T_\ell \subseteq \mathbf{T}$ the set of all terms that Sim has received from Π (during processing of output nodes) prior to sending m_ℓ . Then we have $T_\ell \vdash m_\ell$.

Definition 42 (Indistinguishable simulator) A simulator Sim is indistinguishable for \mathbf{M} , a probabilistic CoSP protocol Π , a computational implementation A , an adversary Adv , and a polynomial p , if

$$\text{Nodes}_{\mathbf{M}, A, \Pi, \text{Adv}}^p(k) \stackrel{c}{\approx} H\text{-Nodes}_{\mathbf{M}, \Pi, \text{Sim}}(k),$$

i.e., if the computational node trace and the hybrid node trace are computationally indistinguishable.

Theorem 7 (Indistinguishable DY-style simulators imply soundness) Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be a symbolic model, let P be a class of CoSP protocols, and let A be a computational implementation of \mathbf{M} . Assume that for every efficient probabilistic CoSP protocol Π (whose corresponding CoSP protocol is in P), every probabilistic polynomial-time adversary Adv , and every polynomial p , there exists a simulator that is indistinguishable and Dolev-Yao style for \mathbf{M} , A , Π , Adv , and p .

Then A is computationally sound for protocols in P .

8 Computational soundness of public key encryption

8.1 The model

We now investigate the computational soundness (in the active case) of IND-CCA secure public key encryption. We use the following symbolic model \mathbf{M}_{pke} :

- Constructors: $\mathbf{C} := \{enc/3, ek/1, dk/1, garbage/1, garbageEnc/2\}$.⁷
- The nonces $\mathbf{N} = \mathbf{N}^P \cup \mathbf{N}^A$ are standard.
- Message type:

$$\mathbf{T} ::= enc(ek(\mathbf{N}), \mathbf{T}, \mathbf{N}) | ek(\mathbf{N}) | dk(\mathbf{N}) | garbage(\mathbf{N}) | garbageEnc(\mathbf{T}, \mathbf{N}).$$

⁷Normally, we would at least add pairs to this. But for simplicity, we restrict ourselves to the bare minimum here.

- Destructors: $\mathbf{D} := \{dec/2, ekof/1, equals/2\}$.⁸ The behavior of the destructors is given by the following rules; an application matching none of these rules evaluates to \perp :

$$\begin{aligned} dec(dk(t_1), enc(ek(t_1), m, t_2)) &= m \\ ekof(enc(ek(t_1), m, t_2)) &= ek(t_1) \\ ekof(garbageEnc(t_1, t_2)) &= t_1 \\ equals(x, x) &= x \end{aligned}$$

- The deduction relation \vdash is standard.

Definition 43 (Key-safe protocols) We call a CoSP protocol Π key-safe if the following holds:

1. The argument of every ek -, dk -computation node and the third argument of every enc -computation node is an N -computation node with $N \in \mathbf{N}^P$. (Here and in the following, we call the nodes referenced by a protocol node its arguments.) We call these N -computation nodes randomness nodes. Two randomness nodes on the same path are annotated with different nonces.
2. Every computation node that is the argument of an ek -computation node or of a dk -computation node on some path p occurs only as argument to ek - and dk -computation nodes on that path p .
3. Every computation node that is the third argument of an enc -computation node on some path p occurs exactly once as an argument in that path p .
4. Every dk -computation node occurs only as the first argument of a dec -destructor node.
5. The first argument of a dec -destructor node is a dk -computation node.
6. There are no computation nodes with the constructors $garbage$, $garbageEnc$, or $N \in \mathbf{N}^A$.

Key-safeness in particular implies that all randomness is fresh and that no key-cycles occur.

Definition 44 (Implementation conditions) We define the following conditions on a computational implementation A :

1. There are disjoint and efficiently recognizable sets of bitstrings representing the types nonces, ciphertexts, encryption keys and decryption keys. The set of all bitstrings of type nonce we denote Nonces_k .⁹ (Here and in the following, k denotes the security parameter.)
2. The functions A_{enc} , A_{ek} , and A_{dk} are length-regular. All nonces $r \in \text{Nonces}_k$ have the same length.
3. A_N for $N \in \mathbf{N}$ returns a uniformly random $r \in \text{Nonces}_k$.

⁸Again, we strive for minimality. We could add a lot of destructors for various tests, such as *isenc* for testing whether a certain term is a ciphertext.

⁹This would typically be the set of all k -bit strings with a tag denoting nonces.

4. Every image of A_{enc} is of type ciphertext, every image of A_{ek} and A_{ekof} is of type encryption key, every image of A_{dk} is of type decryption key.
5. $A_{ekof}(A_{enc}(p, x, y)) = p$ for all p of type encryption key, $x \in \{0, 1\}^*$, $y \in \text{Nonces}_k$.
 $A_{ekof}(e) \neq \perp$ for any e of type ciphertext and $A_{ekof}(e) = \perp$ for any e that is not of type ciphertext.
6. $A_{enc}(p, m, y) = \perp$ if p is not of type encryption key.
7. $A_{dec}(A_{dk}(r), m) = \perp$ if both $r \in \text{Nonces}_k$ and $A_{ekof}(m) \neq A_{ek}(r)$. (This implies that the encryption key is uniquely determined by the decryption key.)
8. $A_{dec}(A_{dk}(r), A_{enc}(A_{ek}(r), m, r')) = m$ for all $r, r' \in \text{Nonces}_k$.
9. We define an encryption scheme (K, E, D) as follows: K picks a random $r \leftarrow \text{Nonces}_k$ and returns $(A_{ek}(r), A_{dk}(r))$. $E(p, m)$ picks a random $r \leftarrow \text{Nonces}_k$ and returns $A_{enc}(p, m, r)$. $D(d, c)$ returns $A_{dec}(d, c)$. We require that then (K, E, D) is IND-CCA secure.
10. For all e of type encryption key and all $m \in \{0, 1\}^*$, the probability that $A_{enc}(e, m, r) = A_{enc}(e, m, r')$ for uniformly chosen $r, r' \in \text{Nonces}_k$ is negligible.

Note that any IND-CCA secure encryption scheme¹⁰ can be transformed into an implementation satisfying the above conditions by suitably tagging and padding the ciphertexts and keys.

8.2 The simulator

In the following, we define distinct nonces $N^m \in \mathbf{N}^A$ for each $m \in \{0, 1\}^*$.

For a protocol Π , an adversary Adv and a polynomial p , we construct the simulator Sim as follows: In the first activation, it chooses $r_N \in \text{Nonces}_k$ for every $N \in \mathbf{N}^P$. At any point in the execution, \mathcal{N} denotes the set of all nonces $N \in \mathbf{N}^P$ that occurred in terms received from Π . \mathcal{R} denotes the set of *randomness nonces* (i.e., the nonces associated with all randomness nodes of Π passed through up to that point).

Sim internally simulates the adversary Adv . When receiving a term $\tilde{t} \in \mathbf{T}$ from Π , it passes $\beta(\tilde{t})$ to Adv where the partial function $\beta : \mathbf{T} \rightarrow \{0, 1\}^*$ is defined below. When Adv answers with $m \in \{0, 1\}^*$, the simulator sends $\tau(m)$ to Π where the function $\tau : \{0, 1\}^* \rightarrow \mathbf{T}$ is defined below. The bitstrings sent from the protocol at control nodes are passed through to Adv and vice versa.

When the simulator receives (info, ν, t) , the simulator increases len by $\ell(t) + 1$ where $\ell : \mathbf{T} \rightarrow \{0, 1\}^*$ is defined below. If $\text{len} > p(k)$, the simulator terminates, otherwise it answers with *proceed*.

Translation functions. The partial function $\beta : \mathbf{T} \rightarrow \{0, 1\}^*$ is defined as follows (where the first matching rule is taken):

- $\beta(N) := r_N$ if $N \in \mathcal{N}$.
- $\beta(N^m) := m$.

¹⁰Here we also assume that decryption fails with probability 0. Alternatively, one can also define encryption schemes where decryption may fail with negligible probability.

- $\beta(\text{enc}(ek(t_1), t_2, M)) := A_{\text{enc}}(\beta(ek(t_1)), \beta(t_2), r_M)$ if $M \in \mathcal{N}$.
- $\beta(\text{enc}(ek(M), t, N^m)) := m$ if $M \in \mathcal{N}$.
- $\beta(ek(N)) := A_{ek}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(ek(N^m)) := m$.
- $\beta(dk(N)) := A_{dk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(\text{garbage}(N^c)) := c$.
- $\beta(\text{garbageEnc}(t, N^c)) := c$.
- $\beta(t) := \perp$ in all other cases.

The total function $\tau : \{0, 1\}^* \rightarrow \mathbf{T}$ is defined as follows (where the first matching rule is taken):

- $\tau(r) := N$ if $r = r_N$ for some $N \in \mathcal{N} \setminus \mathcal{R}$.
- $\tau(r) := N^r$ if r is of type nonce.
- $\tau(c) := \text{enc}(ek(M), t, N)$ if c has earlier been output by $\beta(\text{enc}(ek(M), t, N))$ for some $M \in \mathbf{N}$, $N \in \mathcal{N}$.
- $\tau(c) := \text{enc}(ek(N), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{\text{ekof}}(c)) = ek(N)$ for some $N \in \mathcal{N}$ and $m := A_{\text{dec}}(A_{dk}(r_N), c) \neq \perp$.
- $\tau(e) := ek(N)$ if e has earlier been output by $\beta(ek(N))$ for some $N \in \mathcal{N}$.
- $\tau(e) := ek(N^e)$ if e is of type encryption key.
- $\tau(c) := \text{garbageEnc}(\tau(A_{\text{ekof}}(c)), N^c)$ if c is of type ciphertext.
- $\tau(m) := \text{garbage}(N^m)$ otherwise.

The function $\ell : \mathbf{T} \rightarrow \{0, 1\}^*$ is defined as $\ell(t) := |\beta(t)|$. Note that $\ell(t)$ does not depend on the actual values of r_N because of the length-regularity of A_{enc} , A_{ek} , A_{dk} . Hence $\ell(t)$ can be computed without accessing r_N .

Lecture on
2009-12-10

8.3 Properties of the simulator

Lemma 5 (Indistinguishability) *Assume that Π is a key-safe protocol, and that A satisfies the implementation conditions from Definition 44. Assume a probabilistic polynomial-time adversary Adv and a polynomial p . Then the simulator from Section 8.2 is indistinguishable in the sense of Definition 42.*

Lecture on
2009-12-15

Lemma 6 (Dolev-Yao style simulator) *Assume that Π is an efficient key-safe protocol, and that A satisfies the implementation conditions from Definition 44. Assume a probabilistic polynomial-time adversary Adv and a polynomial p . Then the simulator from Section 8.2 is Dolev-Yao style in the sense of Definition 41.*

Theorem 8 (Computational soundness) *Let P be the set of all key-safe CoSP protocols. Let A be a computational implementation of \mathbf{M}_{pke} that satisfies the implementation conditions from Definition 44.*

Then A is computationally sound for protocols in P .

Lecture on
2010-01-05

9 Automated game-based proofs

An overview over the technique of game-based proofs in cryptography can be found in [Sho04, BR06]. Such proofs constitute a well-structured way of writing cryptographic proofs (directly in the computational model).

Currently, there is only one tool that attempts to automatically find game-based proofs, namely CryptoVerif [Bla06].

9.1 The calculus used in CryptoVerif

We present a simplified version of the calculus used by CryptoVerif. We omit the (very important) concept of arrays.

All definitions and statements in this section are highly simplified. For authoritative information and full detail, see [Bla06].

A *type* is a symbol T together with an interpretation $I_k(T)$ (a family of sets). For each security parameter $k \in \mathbb{N}$, $I_k(T)$ is a subset of $\{0, 1\}^* \cup \{\perp\}$. We require that $I_k(T)$ is efficiently decidable. Examples for types are the type *bitstring* with $I_k(\text{bitstring}) := \{0, 1\}^*$, and *bool* with $I_k(\text{bool}) := \{0, 1\}$ (where 0 is interpreted as false and 1 as true).

A *function* of arity n is a symbol f together with types T_1, \dots, T_n, T (we write $f : T_1 \times \dots \times T_n \rightarrow T$) and with an interpretation $I_k(f)$. For each security parameter k , $I_k(f)$ is a function from $I_k(T_1) \times \dots \times I_k(T_n)$ to $I_k(T)$. We require that $I_k(f)$ is efficiently computable.

A *term* obeys the following grammar:

$$M ::= x \mid f(M_1, \dots, M_n)$$

Here x stands for a variable name, and f for a function symbol.

We distinguish two types of *processes*: *input processes* and *output processes*. The input processes obey the following grammar:

$$\begin{array}{ll}
 Q ::= & \\
 0 & \text{empty process} \\
 Q|Q' & \text{parallel composition} \\
 !^{i \leq m} Q & \text{parallel composition of } m \text{ copies of } Q \\
 \text{newChannel } c; Q & \text{allocation of a new channel} \\
 c(x_1 : T_1, \dots, x_n : T_n); P & \text{input on channel } c \text{ into variables } x_1, \dots, x_n
 \end{array}$$

Here m stands for a *parameter*, i.e., an efficiently computable polynomially-bounded function of the security parameter. P stands for an output process.

Output processes obey the following grammar:

$P ::=$	
$\bar{c}\langle M_1, \dots, M_n \rangle; Q$	output on channel c of values M_1, \dots, M_n
$\text{new } x : T; P$	assignment of random value of type T to x
$\text{let } x : T = M \text{ in } P$	compute M and assign to x
$\text{if } M \text{ then } P \text{ else } P'$	conditional (M must evaluate to <i>bool</i>)

Well-typed processes. [Bla06] imposes certain syntactical and well-typedness conditions on processes. These guarantee for example that no undefined variables are accessed, and that no functions are applied to values of the wrong type. We omit a description of these conditions here and refer to [Bla06].

Semantics (informal). We sketch the semantics of the calculus on an informal level. The execution of an input process Q starts with the process $Q|\overline{start}\langle \rangle$. ($\overline{start}\langle \rangle$ sends an empty message on channel $start$ to get the input process Q a chance to start running.) Then, in each step, we probabilistically reduce the current process according to the following rules:

- A process $\text{new } x : T; P$ is reduced to P , and x is assigned a uniformly random value from $I_k(T)$.
- A process $\text{let } x : T = M \text{ in } P$ is reduced to P , and x is assigned the result of evaluating M (using the interpretations $I_k(f)$ of the function symbols f occurring in M).
- A process $\text{if } M \text{ then } P \text{ else } P'$ is reduced to P or P' , depending on whether M evaluates to 1 or 0.
- A process $!^{i \leq m} Q$ is reduced to $Q | \dots | Q$ ($n(k)$ copies).
- A process $\text{newChannel } c; Q$ is reduced to Q , where each occurrence of c in Q is replaced by c' , where c' is a fresh channel name.
- The process 0 (if parallel composed with some other process) is discarded. If the current process is 0 (and there is no other process parallel composed with it), the execution stops.
- If $c(x_1 : T_1, \dots, x_n : T_n); P$ and $\bar{c}\langle M_1, \dots, M_n \rangle; Q$ are composed in parallel, and the terms M_1, \dots, M_n evaluate to values $b_1, \dots, b_n \in I_k(T_1), \dots, I_k(T_n)$, then the processes reduce to P and Q , and the variables x_1, \dots, x_n are assigned the values b_1, \dots, b_n . If there are several processes $c(x_1 : T_1, \dots, x_n : T_n); P$ satisfying these conditions, one is chosen uniformly at random. (The syntactical conditions on processes ensure that there is at most one process of the form $\bar{c}\langle M_1, \dots, M_n \rangle; Q$.)

- Note that a process may access variables that are bound by a let- or new-statement in another (parallel-composed) process. For example, if we have $Q' | \text{new } x : T; Q$, then Q' might access x . In particular, if the adversary is a process running in parallel with a process Q , then the adversary can access all variables of Q unless we put explicit restrictions on the variables occurring in the adversary.

For a formal account of the semantics, see [Bla06].

In order to model events, we use channels. For example, raising an event *bad* can be modeled by the process $\overline{\text{bad}}\langle \rangle; Q$. We write $\Pr[Q \rightarrow_k \bar{c}\langle a \rangle]$ to denote the probability that Q will eventually output the value a on channel c .

Definition 45 (Observational equivalence) *An evaluation context is a context of the grammar*

$$C ::= \square \mid \text{newChannel } c; C \mid (Q|C) \mid (C|Q).$$

Given two input processes Q_L and Q_R and a set of variables V , we say that an evaluation context C is acceptable for Q_L, Q_R, V iff $\text{vars}(C) \cap \text{vars}(Q_L, Q_R) \subseteq V$ (where $\text{vars}(C)$ denotes all variables in C , even those bound by a let- or new-statement).

We say Q_L and Q_R are observationally equivalent with public variables V (written $Q_L \approx^V Q_R$) iff for all evaluation contexts acceptable for Q_L, Q_R, V , for all channels c , and for all bitstrings a , we have that

$$\left| \Pr[C[Q_L] \rightarrow_k \bar{c}\langle a \rangle] - \Pr[C[Q_R] \rightarrow_k \bar{c}\langle a \rangle] \right|$$

is negligible in k .

If $V = \emptyset$, we write $Q_L \approx Q_R$ instead of $Q_L \approx^V Q_R$.

9.2 Game transformations

The basic approach of CryptoVerif is to start with some initial process Q and to automatically apply transformations that change Q into an observationally equivalent process. The final process Q^* is (hopefully) of a form such that the security property that is to be shown can immediately be seen to hold. E.g., if the goal is to show that Q does not raise an event *bad*, and Q^* does not contain any instruction for raising the event *bad*, then Q^* raises the event *bad* with probability 0, and thus, because $Q \approx Q^*$, Q raises the event *bad* with negligible probability.

There are two main transformations in CryptoVerif: Simplification, and crypto transformations.

9.2.1 Simplification

The simplification algorithm is parametrized over a set of user-defined equalities of the form

$$\forall x_1 : T_1, \dots, \forall x_m : T_m. M$$

The meaning of each such equality is that the following holds: For all security parameters k and all environments E (mapping variables to bitstrings), if $E(x_1) \in$

$I_k(T_1), \dots, E(x_m) \in I_k(T_m)$, then $E, M \Downarrow \text{true}$. Here *true* formally means the bit-string 1, and $E, M \Downarrow b$ means that M evaluates to b (using the interpretations $I_k(f)$ of the functions f occurring in the term M).

The simplification algorithm then does – roughly – the following:

- It applies certain built-in rewrite rules to the terms occurring in the current process Q . These built-in rules comprise rules like $\text{not}(x = y) \rightarrow x \neq y$ or $x \vee \text{true} \rightarrow x$.
- It applies rewrite rules corresponding to the user-defined equalities. For each equality $\forall x_1 : T_1, \dots, \forall x_m : T_m. M$, we distinguish the following cases:
 - If $M = (M_1 = M_2)$, then use the rewrite rule $M_1 \rightarrow M_2$.
 - If $M = (M_1 \neq M_2)$, then use the rewrite rules $(M_1 = M_2) \rightarrow \text{false}$ and $(M_1 \neq M_2) \rightarrow \text{true}$.
 - Otherwise, use the rewrite rule $M \rightarrow \text{true}$.
- For each position of the protocol, derive true facts that are guaranteed to hold at that position. For example, in the then-branch of *if M then P*, the fact M necessarily holds. Use these true facts as additional rewrite rules.
- Remove useless variables: If $x \notin \text{vars}(Q) \cup V$, use the rewrite rules *let x : T = M in P* $\rightarrow P$ and *new x : T in P* $\rightarrow P$.
- Apply all the above until a fixpoint is reached.

9.2.2 Crypto transformations

In order to use crypto transformations in CryptoVerif, the user first needs to introduce cryptographic assumptions. Such an assumption is encoded as an equivalence of the form $(G_1, \dots, G_m) \approx (G'_1, \dots, G'_m)$ where the G_i, G'_i are of the following grammar:

$$\begin{aligned}
 G &::= !^{i \leq n} \text{new } y_1 : T_1; \dots; \text{new } y_l : T_l : (G_1, \dots, G_m) \\
 &\quad (x_1 : T_1, \dots, x_l : T_l) \rightarrow FP \\
 FP &::= M \\
 &\quad \text{new } x : T; FP \\
 &\quad \text{let } x : T = M \text{ in } FP \\
 &\quad \text{if } M \text{ then } FP \text{ else } FP
 \end{aligned}$$

Each such group of functions (G_1, \dots, G_m) represents some process $\llbracket(G_1, \dots, G_m)\rrbracket$ that implements the functions G_1, \dots, G_m . That is

$$\begin{aligned} \llbracket(G_1, \dots, G_m)\rrbracket &:= \llbracket G_1 \rrbracket | \dots | \llbracket G_m \rrbracket \\ \llbracket(x_1 : T_1, \dots, x_l : T_l) \rightarrow FP\rrbracket &:= c(x_1 : T_1, \dots, x_l : T_l); \llbracket FP \rrbracket \\ \llbracket!^{i \leq n} \text{ new } y_1 : T_1; \dots : (G_1, \dots, G_m)\rrbracket &:= !^{i \leq n} c(); \text{ new } y_1 : T_1; \dots ; \bar{c}\langle \rrbracket(G_1, \dots, G_m)\rrbracket \\ \llbracket M \rrbracket &:= \bar{c}\langle M \rangle \\ \llbracket \text{ new } x : T; FP \rrbracket &:= \text{ new } x : T; \llbracket FP \rrbracket \\ \llbracket \text{ let } x : T = M \text{ in } FP \rrbracket &:= \text{ let } x : T = M \text{ in } \llbracket FP \rrbracket \\ \llbracket \text{ if } M \text{ then } FP_1 \text{ else } FP_2 \rrbracket &:= \text{ if } M \text{ then } \llbracket FP_1 \rrbracket \text{ else } \llbracket FP_2 \rrbracket \end{aligned}$$

Here c stands for an arbitrary channel name. All occurrences of c are mapped to difference channel names.

Then $(G_1, \dots, G_m) \approx (G'_1, \dots, G'_m)$ denotes $\llbracket(G_1, \dots, G_m)\rrbracket \approx \llbracket(G'_1, \dots, G'_m)\rrbracket$. It is the user's responsibility to check that the observational equivalence $\llbracket(G_1, \dots, G_m)\rrbracket \approx \llbracket(G'_1, \dots, G'_m)\rrbracket$ indeed holds.

When CryptoVerif tries to apply a crypto transformation parametrized by an equivalence $LHS \approx RHS$ to a game Q , it tries to identify variables x introduced by *new* in Q with variables in LHS . Then, CryptoVerif tries to find subterms M in Q that can be computed by communicating with the process $\llbracket LHS \rrbracket$ instead of computing them directly in Q (e.g., if LHS contains $(x : T) \rightarrow f(x)$, then $f(x)$ in Q can be replaced by sending x to $\llbracket LHS \rrbracket$ and receiving $f(x)$ from $\llbracket LHS \rrbracket$). We get a modified process $Q' | \llbracket LHS \rrbracket$, and CryptoVerif ensures that $Q \approx Q' | \llbracket LHS \rrbracket$. Then, since $\llbracket LHS \rrbracket \approx \llbracket RHS \rrbracket$, we have $Q' | \llbracket LHS \rrbracket \approx Q' | \llbracket RHS \rrbracket$. Then, in Q' , we replace all communications with $\llbracket RHS \rrbracket$ by direct computations of the corresponding terms. We call the resulting process Q'' . We then have $Q' | \llbracket RHS \rrbracket \approx Q''$. Summarizing, $Q \approx Q''$. The crypto transformation then replaces the current process Q by the new process Q'' .

Lecture on
2010-01-19

10 Computational soundness of zero-knowledge proofs

Warning: The following material has to be taken with some care. The original result on computational soundness of zero-knowledge proofs [BU10] has been derived using a different formalism for modeling protocols than the CoSP framework used in this lecture. Since I did not wish to introduce yet another formalism, I have “translated” the results from [BU10] into the CoSP framework. I have guessed how the results *should* look like using CoSP, but since I have not redone the proofs, there is no guarantee that the results as formalized below are completely correct. I am, however, quite confident that a formalization of the results from [BU10] in CoSP would be very similar to what I present below.

Examples for papers that use a symbolic modeling of zero-knowledge: [BMU08, BHM08a, BHM08b].

10.1 Symbolic model

A *ZK-term* is defined by the following grammar:

$$\text{ZKterm} ::= \alpha_i | \beta_i | \text{enc}(\text{ZKterm}, \text{ZKterm}, \alpha_i) | \text{dec}(\alpha_i, \text{ZKterm}).$$

The interpretation of α_i and β_i is as a placeholder for the i -th component of the witness and the public part, respectively.

A ZK-atom is of the form

$$\text{ZKatom} ::= \text{ZKterm} = \text{ZKterm}.$$

The interpretation of a ZK-atom $L = R$ is that $L = R$ holds after substituting the α_i and β_i by the witness and public part of some ZK-proof.

A ZK-formula is a Boolean formula over ZK-atoms. I.e., if B is a Boolean formula,

$$\text{ZKformula} ::= B(\text{ZKatom}, \dots, \text{ZKatom}).$$

The largest index i such that α_i occurs in a ZK-formula F we call the α -arity α_F . The largest index i such that β_i occurs in a ZK-formula F we call the β -arity β_F .

For a ZK-formula F , we denote by $\text{atoms}(F)$ the set of all ZK-atoms in F .

Definition 46 (True proofs) *Let F be a ZK-formula. Fix sets of constructors \mathbf{C} and nonces \mathbf{N} , a set of terms \mathbf{T}_0 , and a set of destructors \mathbf{D} . Let a term of the form $zk_F(a_1, \dots, a_{\alpha_F}; b_1, \dots, b_{\beta_F}; R)$ be given. (Here a_1, \dots, a_n represent the witness and b_1, \dots, b_n the public part of the ZK-proof. R is the randomness used for constructing the ZK-proof.)*

For $C/n \in \mathbf{C} \cup \mathbf{D}$, let $\text{eval}(C(t_1, \dots, t_n)) := C(\text{eval}(t_1), \dots, \text{eval}(t_n))$ if $C(\text{eval}(t_1), \dots, \text{eval}(t_n)) \in \mathbf{T}_0$ and $\text{eval}(C(t_1, \dots, t_n)) := \perp$ otherwise. For $D/n \in \mathbf{D}$, let $\text{eval}(D(t_1, \dots, t_n)) := D(\text{eval}(t_1), \dots, \text{eval}(t_n))$ (the result of evaluating the function D).

Let $\sigma := \{a_1/\alpha_1, \dots, a_{\alpha_F}/\alpha_{\alpha_F}, b_1/\beta_1, \dots, b_{\beta_F}/\beta_{\beta_F}\}$ be the substitution that replaces all α_i by a_i and all β_i by b_i .

For an atom $(l = r) \in \text{atoms}(F)$, we say that $(l = r)$ is *valid* iff $\text{eval}(l\sigma) \neq \perp$ and $\text{eval}(r\sigma) \neq \perp$. We say $(l = r)$ is *true* iff $(l = r)$ is valid and $\text{eval}(l\sigma) = \text{eval}(r\sigma)$.

Let $F =: B(a_1, \dots, a_n)$ where B is a Boolean formula and $\{a_1, \dots, a_n\} = \text{atoms}(F)$. Set $\tilde{a}_i := \text{true}$ if a_i is true and $\tilde{a}_i := \text{false}$ otherwise.

We call $zk_F(a_1, \dots, a_{\alpha_F}; b_1, \dots, b_{\beta_F}; R)$ a *true ZK-proof* iff all $t \in \text{atoms}(F)$ are valid and $B(\tilde{a}_1, \dots, \tilde{a}_n) = \text{true}$ holds (when B is evaluated as a Boolean expression).

Let \mathcal{F} be a finite set of ZK-formulas.

We define the following constructors:

$$\mathbf{C} := \{ \text{enc}/3, \text{ek}/1, \text{dk}/1, \text{garbage}/1, \text{garbageEnc}/2, \\ zk_F/(\alpha_F + \beta_F + 1), \text{garbageZK}_F/(\beta_F + 1) : F \in \mathcal{F} \}$$

To increase readability, we use semicolons instead of commas to separate witness (first α_F arguments), public part (next β_F arguments) and randomness (last argument) of a zk_F -term. Similarly for garbageZK_F .

Let the nonces \mathbf{N} be standard.

We define the following message type \mathbf{T}_0 :

$$\begin{aligned} \mathbf{T}_0 ::= & \text{enc}(ek(\mathbf{N}), \mathbf{T}_0, \mathbf{N}) | ek(\mathbf{N}) | dk(\mathbf{N}) | \text{garbage}(\mathbf{N}) | \text{garbageEnc}(\mathbf{T}_0, \mathbf{N}) \\ & | zk_F(\mathbf{T}_0, \dots, \mathbf{T}_0; \mathbf{T}_0, \dots, \mathbf{T}_0; \mathbf{N}) | \text{garbageZK}_F(\mathbf{T}_0, \dots, \mathbf{T}_0; \mathbf{N}) \end{aligned}$$

(The message type \mathbf{T}_0 is only used as an intermediate definition in order to be able to apply Definition 46, which in turn is needed to define the message type \mathbf{T} below.)

The destructors \mathbf{D} are:

$$\mathbf{D} := \{ \text{dec}/2, \text{ekof}/1, \text{equals}/2, \text{zkverify}_F/1, \text{zkpublic}_{F,i}/1 : F \in \mathcal{F}, i = 1, \dots, \beta_F \}$$

The constructors dec , ekof , equals are defined as in Section 8.1. The remaining destructors are defined by the following rules:

$$\begin{aligned} \text{zkverify}_F(\text{zk}_F(a_1, \dots, a_n; b_1, \dots, b_m; r)) &= \text{zk}_F(a_1, \dots, a_n; b_1, \dots, b_m; r) \\ \text{zkpublic}_{F,i}(\text{zk}_F(a_1, \dots, a_n; b_1, \dots, b_m; r)) &= b_i \\ \text{zkpublic}_{F,i}(\text{garbageZK}_F(b_1, \dots, b_m; r)) &= b_i. \end{aligned}$$

We define the message type \mathbf{T} as the set of all terms $t \in \mathbf{T}_0$ such that every $\text{zk}_F(\dots)$ -subterm of t is a true ZK-proof in the sense of Definition 46.

The deduction relation \vdash is standard.

Definition 47 (Symbolic model for ZK) Let $\mathbf{M}_{zk}^{\mathcal{F}} := (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$. We call $\mathbf{M}_{zk}^{\mathcal{F}}$ the symbolic model for ZK with formulas \mathcal{F} .

Note that the restriction of \mathbf{T} to messages containing only true proofs has the effect that the adversary can only derive true proofs, even though the definition of \vdash does not explicitly mention true proofs.

Lecture on
2010-01-28

10.2 Computational implementation

We state the following definition not for an arbitrary relation R , but directly specialized to the relation R with CRw iff $C(w) = 1$. This is sufficient for our purposes.

Definition 48 (Symbolically-sound zero-knowledge proof system) A symbolically-sound zero-knowledge proof system is a tuple of polynomial-time algorithms (K, P, V, S, E) with the following properties:

- **Completeness:** Let a polynomial-time adversary Adv be given. Let $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow K(1^k)$. Let $(C, w) \leftarrow \text{Adv}(1^k, \text{crs})$. Let $\pi \leftarrow P(1^k, C, w, \text{crs})$. Then with overwhelming probability in k , $C(w) = 0$ or $V(1^k, C, \pi, \text{crs}) = 1$.
- **Extractability:** Let a polynomial-time adversary Adv be given. Let $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow K(1^k)$. Let $(C, \pi) \leftarrow \text{Adv}(1^k, \text{crs})$. Let $w \leftarrow E(1^k, C, \pi, \text{extd})$. Then with overwhelming probability, if $V(1^k, C, \pi, \text{crs}) = 1$ then $C(w) = 1$.

- **Unpredictability:** Let a polynomial-time adversary Adv be given. Let $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow K(1^k)$. Let $(C, w, \pi') \leftarrow \text{Adv}(1^k, \text{crs}, \text{simtd}, \text{extd})$. Then with overwhelming probability, we have $\pi' \neq P(1^k, C, w, \text{crs})$.
- **Extraction zero-knowledge:** Let a polynomial-time adversary Adv be given. Consider the following experiment parametrized by a bit b : Let $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow K(1^k)$. Let $(C, w, \text{state}) \leftarrow \text{Adv}^{E(1^k, \cdot, \cdot, \text{extd})}(1^k, \text{crs}, \text{simtd})$. Then let $\pi \leftarrow P(1^k, C, w, \text{crs})$ if $b = 0$ and $\pi \leftarrow S(1^k, C, \text{crs}, \text{simtd})$ if $b = 1$. Let $\text{guess} = \text{Adv}^{E(1^k, \cdot, \cdot, \text{extd})}(1^k, \text{crs}, \text{simtd}, \text{state}, \pi)$. Let $P_b(k)$ denote the following probability:

$$P_b(k) := \Pr[\text{guess} = 1 \text{ and } C(w) = 1 \text{ and } \\ (C, \pi) \text{ has not been queried from } E(1^k, \cdot, \cdot, \text{extd})].$$

Then $|P_0(k) - P_1(k)|$ is negligible.

- **Length-regularity.** Let two circuits C, C' and witnesses w, w' with $C(w) = C'(w') = 1$ be given such that $|C| = |C'|$ and $|w| = |w'|$. Let $(\text{crs}, \text{simtd}, \text{extd}) \leftarrow K(1^k)$. Then let $\pi \leftarrow P(1^k, C, w, \text{crs})$ and $\pi' \leftarrow P(1^k, C', w', \text{crs})$. Then $|\pi| = |\pi'|$ holds with probability 1.
- **Deterministic verification and extraction.** The algorithms V and E are deterministic.

Protocols satisfying these notions have been presented in [GO07, Sect. 4].

Definition 49 (ZK-circuit) Let F be a ZK-formula with α -arity $n = \alpha_F$ and β -arity $m = \beta_F$. Let b_1, \dots, b_m be bitstrings and l_1, \dots, l_n be nonnegative integers. The ZK-circuit $C_F^{b_1, \dots, b_m; l_1, \dots, l_n}$ is a circuit that returns 1 on input (a_1, \dots, a_n) with $|a_i| = l_i$ iff $\text{ceval}_{a_1, \dots, a_n; b_1, \dots, b_m}(F) \neq \perp$.

Here $\text{ceval}_{a_1, \dots, a_n; b_1, \dots, b_m}(F) = \text{ceval}(F)$ is recursively defined as follows:

- If $F = B(t_1, \dots, t_s)$ for a Boolean formula B and ZK-atoms t_1, \dots, t_s , then $\text{ceval}(F) = \text{true}$ iff $v_i := \text{ceval}(t_i) \neq \perp$ for all i and $B(v_1, \dots, v_s) = \text{true}$.
- $\text{ceval}(l = r) = \text{true}$ for a ZK-atom $l = r$ iff $\text{ceval}(l) \neq \perp$ and $\text{ceval}(r) \neq \perp$ and $\text{ceval}(l) = \text{ceval}(r)$.
- $\text{ceval}(l = r) = \text{false}$ for a ZK-atom $l = r$ iff $\text{ceval}(l) \neq \perp$ and $\text{ceval}(r) \neq \perp$ and $\text{ceval}(l) \neq \text{ceval}(r)$.
- $\text{ceval}(l = r) = \perp$ for a ZK-atom $l = r$ iff $\text{ceval}(l) = \perp$ or $\text{ceval}(r) = \perp$.
- $\text{ceval}(f(t_1, \dots, t_s)) = A_f(\text{ceval}(t_1), \dots, \text{ceval}(t_s))$ for a ZK-term $f(t_1, \dots, t_s)$ with constructor or destructor f/s .
- $\text{ceval}(\alpha_i) = a_i$.
- $\text{ceval}(\beta_i) = b_i$.

Definition 50 (Implementation conditions) We define the following conditions on a computational implementation A :

1. A satisfies the implementation conditions from Definition 44.

2. There is an efficiently recognizable set of bitstrings representing the type ZK-proof. This set is disjoint from the sets defined in Definition 44.
3. There is a symbolically-sound encryption scheme (K, P, V, S, E) where P takes a uniformly random element of Nonces_k as randomness.
4. We assume that a value $\text{crs} \leftarrow K(1^n)$ is chosen at the beginning of the protocol.
5. $A_{zk_F}(1^k, a_1, \dots, a_n; b_1, \dots, b_m; r)$ checks whether $C_F^{l_1, \dots, l_n}(a_1, \dots, a_n) = 1$ with $l_i := |a_i|$. If not, A_{zk_F} returns \perp . Otherwise, A_{zk_F} invokes $\pi \leftarrow P(1^k, C_F^{l_1, \dots, l_n}, (a_1, \dots, a_n), \text{crs})$ with randomness r . Then A_{zk_F} returns $\langle \pi, F, b_1, \dots, b_m, l_1, \dots, l_n \rangle$ where $\langle \cdot \rangle$ is a length-regular tuple-encoding that returns bitstrings of type ZK-proof and where $n = \alpha_F$ and $m = \beta_F$.
6. $A_{zk_{\text{public}}_F}(1^k, z) = b_i$ if $z = \langle \pi, F, b_1, \dots, b_m, l_1, \dots, l_n \rangle$ for some b_i, l_i, π (and $n = \alpha_F$ and $m = \beta_F$). Otherwise $A_{zk_{\text{public}}_F}(z) = \perp$.
7. $A_{zk_{\text{verify}}_F}(1^k, z) = b_i$ if $z = \langle \pi, F, b_1, \dots, b_m, l_1, \dots, l_n \rangle$ for some b_i, l_i, π (and $n = \alpha_F$ and $m = \beta_F$) and $V(1^k, C_F^{l_1, \dots, l_n}, z, \text{crs}) = 1$. Otherwise $A_{zk_{\text{verify}}_F}(z) = \perp$.

Note that here, somewhat simplifying, we have assumed that a CRS crs is chosen at the beginning of the protocol as $\text{crs} \leftarrow K(1^k)$ and given as argument to the algorithms P and V . Alternatively, one could explicitly model the CRS in the symbolic model: Then the constructor zk_F and the destructor zk_{verify}_F (and their corresponding computational implementation) would have an additional argument taking the CRS. We do not pursue this approach further at this point.

Lecture on
2010-02-02

10.3 ZK-safe protocols

Definition 51 (ZK-safe protocols) We call a CoSP protocol Π ZK-safe if the following holds:

1. The argument of every ek -, dk -computation node and the third argument of every enc -computation node and the last argument of every zk_F -computation node is an N -computation node with $N \in \mathbf{N}^P$. (Here and in the following, we call the nodes referenced by a protocol node its arguments.) We call these N -computation nodes randomness nodes. Two randomness nodes on the same path are annotated with different nonces.
2. If $enc(\text{ZKterm}, \text{ZKterm}, \alpha_i)$ occurs in F , then the i -th argument of a zk_F -computation node is a randomness node.
3. Every computation node that is the argument of an ek -computation node or of a dk -computation node on some path p occurs only as argument to ek - and dk -computation nodes on that path p .
4. Every computation node ν' that is the third argument of an enc -computation node on some path p occurs only as the third-argument to only one enc -computation node ν and as the i -th argument to a zk_F -computation node such that α_i occurs in F only within a ZK-term $enc(\text{ZKterm}, \text{ZKterm}, \alpha_i)$ that will be evaluated to the same term as the enc -constructor of ν .
5. Every dk -computation node occurs only as the first argument of a dec -destructor

node and as the i -th argument to a zk_F -computation node such that α_i occurs in F only within a ZK-term $dec(\alpha_i, \text{ZKterm})$.

6. The first argument of a dec -destructor node is a dk -computation node.
7. If $dec(\alpha_i, \text{ZKterm})$ occurs in F , then the i -th argument of a zk_F -computation node is a dk -computation node.
8. There are no computation nodes with the constructors $garbage$, $garbageEnc$, $garbageZK$, or $N \in \mathbf{N}^A$.

Note: The (somewhat involved) requirements on the uses of randomness-nodes and dk -computation nodes ensure that if we would replace each zero-knowledge proof by a subtree that checks whether the proof is true (whether the witness matches statement and formula), then we would get a key-safe CoSP protocol. This is because the computational implementation A_{zk_F} checks whether $C_F^{l_1, \dots, l_n; b_1, \dots, b_n}(a_1, \dots, a_n) = 1$ before producing the ZK-proof. Evaluating this circuit is equivalent to checking whether the ZK-proof is true.

10.4 Computational soundness proof

We construct a simulator Sim as in Section 8.2, except that we add the following rules to the definitions of the translation functions β and τ :

- $\beta(zk_F(a_1, \dots, a_n; b_1, \dots, b_m, M)) := A_{zk_F}(\beta(a_1), \dots, \beta(a_n); \beta(b_1), \dots, \beta(b_m); r_M)$ if $M \in \mathcal{N}$.
- $\beta(zk_F(a_1, \dots, a_n; b_1, \dots, b_m, N^m)) := m$.
- $\beta(garbageZK_F(b_1, \dots, b_m, N^m)) := m$.

and

- $\tau(z) := zk_F(a_1, \dots, a_n; b_1, \dots, b_m; M)$ if z has earlier been output by $\beta(zk_F(a_1, \dots, a_n; b_1, \dots, b_m; M))$.
- $\tau(z) := zk_F(\tau(a_1), \dots, \tau(a_n); \tau(b_1), \dots, \tau(b_m); M^z)$ if z is of type zero-knowledge and $z = \langle \pi, F, b_1, \dots, b_m, l_1, \dots, l_n \rangle$ such that $V(1^k, C_F^{l_1, \dots, l_n; b_1, \dots, b_n}, \pi, crs) = 1$ and $(a_1, \dots, a_n) := E(1^k, C_F^{l_1, \dots, l_n; b_1, \dots, b_n}, \pi, extd) = 1$. (If E fails or $C_F^{l_1, \dots, l_n; b_1, \dots, b_n}(a_1, \dots, a_n) \neq 1$, we say a ZK-break occurred.)
- $\tau(z) := garbageZK_F(\tau(b_1), \dots, \tau(b_m); M^z)$ if z is of type zero-knowledge and $z = \langle \pi, F, b_1, \dots, b_m, l_1, \dots, l_n \rangle$ such that $V(1^k, C_F^{l_1, \dots, l_n; b_1, \dots, b_n}, \pi, crs) \neq 1$.

Lemma 7 (Indistinguishability) *Assume that Π is a ZK-safe protocol, and that A satisfies the implementation conditions from Definition 50. Assume a probabilistic polynomial-time adversary Adv and a polynomial p . Then the simulator Sim specified in this section is indistinguishable in the sense of Definition 42.*

Lemma 8 (Dolev-Yao style simulator) *Assume that Π is an efficient key-safe protocol, and that A satisfies the implementation conditions from Definition 50. Assume a probabilistic polynomial-time adversary Adv and a polynomial p . Then the simulator Sim specified in this section is Dolev-Yao style in the sense of Definition 41.*

Conjecture 9 (Computational soundness) ¹¹ Let P be the set of all ZK-safe CoSP protocols. Let A be a computational implementation of $\mathbf{M}_{zk}^{\mathcal{F}}$ that satisfies the implementation conditions from Definition 50.

Then A is computationally sound for protocols in P .

References

- [AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 104–115, New York, NY, USA, 2001. ACM Press.
- [AR02] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.
- [BCK05] Mathieu Baudet, Véronique Cortier, and Steve Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 652–663. Springer, 2005.
- [BHM08a] Michael Backes, Cătălin Hrițcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *21th IEEE Symposium on Computer Security Foundations (CSF 2008)*, pages 195–209. IEEE Computer Society Press, June 2008.
- [BHM08b] Michael Backes, Cătălin Hrițcu, and Matteo Maffei. Type-checking zero-knowledge. In *15th ACM Conference on Computer and Communications Security (CCS 2008)*, pages 357–370. ACM Press, October 2008.
- [BHU09] Michael Backes, Dennis Hofheinz, and Dominique Unruh. CoSP: A general framework for computational soundness proofs. In *ACM CCS 2009*, pages 66–78. ACM Press, November 2009. Preprint on IACR ePrint 2009/080.
- [Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2006*, pages 140–154. IEEE Computer Society, 2006. Extended version online available at <http://eprint.iacr.org/2005/401.ps>. The tool is available at <http://www.cryptoverif.ens.fr/>.
- [BLMW07] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of ddh with applications to protocol analysis and computational soundness. In Menezes [Men07], pages 482–499.

¹¹I call this a conjecture because I have not checked the proof in the CoSP framework. There is a proof of a similar theorem in another framework in [BU10].

- [BMU08] Michael Backes, Matteo Maffei, and Dominique Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *IEEE Symposium on Security and Privacy, Proceedings of SSP'08*, pages 202–215, May 2008. Preprint on IACR ePrint 2007/289, <http://eprint.iacr.org/2007/289>.
- [BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT 2006*, volume 4004 of *LNCS*. Springer, 2006. Full version online available at <http://eprint.iacr.org/2004/331.ps>.
- [BU10] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs. *Journal of Computer Security*, 2010. To appear. Preprint on IACR ePrint 2008/152, <http://eprint.iacr.org/2008/152>.
- [GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Menezes [Men07], pages 323–341. Full version available at <http://www.cs.ucla.edu/~rafail/PUBLIC/85.pdf>. The definition of extraction zero-knowledge is only contained in the full version.
- [Men07] Alfred Menezes, editor. *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*. Springer, 2007.
- [Pie02] Benjamin C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.
- [Sho04] Victor Shoup. Sequences of games: A tool for taming complexity in security proofs. IACR ePrint Archive, November 2004. Online available at <http://eprint.iacr.org/2004/332.ps>.
- [Unr09] Dominique Unruh. Lecture “Formal Methods and Cryptography”, winter 2009. Webpage is <http://crypto.m2ci.org/teaching/ws09/fmc/>.

Index

- α -arity, 27
- arity
 - α -, 27
 - β -, 27
- β -arity, 27
- bounded, 8
- closed
 - prefix-, 2
- completeness
 - of ZK proofs, 28
- computation node, 14
- computational soundness
 - (CoSP), 16
- computational execution, 15
 - of a p.s.p., 10
- computational implementation, 10
- computational node trace, 15
- computationally satisfy, 16
- constructor, 9
- context, 2
 - evaluation (CryptoVerif), 24
- control node, 14
- CoSP
 - computational soundness, 16
- CoSP protocol
 - probabilistic, 15
- CoSP protocol, 14
 - efficient, 16
- CryptoVerif, 22
- decidable
 - efficiently, 2
- deduction relation
 - standard, 9
- deduction relation, 9
- destructor, 9
- deterministic extraction, 29
- deterministic verification, 29
- Dolev-Yao style, 18
- DY, *see* Dolev-Yao style
- efficient CoSP protocol, 16
- efficiently decidable, 2
- encryption
 - symmetric
 - symbolic model for, 11
- equivalence
 - observational (CryptoVerif), 24
- evaluation context
 - (CryptoVerif), 24
- execution
 - computational, 15
 - computational, of a p.s.p., 10
 - symbolic, 14
- extractability, 28
- extraction
 - deterministic, 29
- extraction zero-knowledge, 29
- failure-free, 12
- fresh randomness, 12
- full hybrid trace, 17
- full trace, 15
 - hybrid, 17
- function
 - (CryptoVerif), 22
- hold computationally
 - passive secrecy property, 11
- hold symbolically
 - passive secrecy property, 10
- hole, 2
- hybrid execution, 17
- hybrid node trace, 18
- hybrid trace
 - full, 17
- implementation
 - computational, 10
- IND-CPA, 12
 - computational implementation, 12
- IND-OT-CPA
 - symmetric, 3

- indistinguishable simulator, 18
- induction over rules, 9
- induction principle, 8
- inference rule, 8
- input node, 14
- input process
 - (CryptoVerif), 22
- key-cycle-free, 11
- key-safe, 19
- length-regular, 11
- length-regularity
 - of ZK proofs, 29
- message type, 9
- model
 - symbolic, 9
 - for symmetric encryption, 11
- monotone, 8
- negligible, 2
- node
 - computation, 14
 - control, 14
 - input, 14
 - nondeterministic, 14
 - output, 14
 - randomness, 19
- node trace, 15
 - computational, 15
 - hybrid, 18
- non-negligible, 2
- nonce, 9
 - randomness, 20
 - unpredictable, 12
- nonces
 - standard, 9
- nondeterministic node, 14
- noticeable, 2
- observational equivalence
 - (CryptoVerif), 24
- output node, 14
- output process
 - (CryptoVerif), 22
- overwhelming, 2
- parameter
 - (CryptoVerif), 22
- passive secrecy property, 10
- prefix-closed, 2
- probabilistic CoSP protocol, 15
- process
 - (CryptoVerif), 22
- proof
 - zero-knowledge, 26
- property
 - passive secrecy, 10
 - trace, 15
- protocol
 - CoSP, 14
 - efficient CoSP, 16
 - probabilistic CoSP, 15
- randomness
 - fresh, 12
- randomness node, 19
- randomness nonce, 20
- relation
 - standard deduction, 9
- rule
 - (inference), 8
- rule induction, 9
- safe
 - key-, 19
 - ZK-, 30
- satisfy
 - computationally, 16
 - symbolically, 15
- secrecy
 - strong, 10
 - weak, 10
- secrecy property
 - secret, 10
- semantics
 - (CryptoVerif), 23
- simulator, 17

- Dolev-Yao style, 18
- indistinguishable, 18
- soundness
 - computational (CoSP), 16
- standard deduction relation, 9
- standard nonces, 9
- substitution, 2
- symbolic model
 - for symmetric encryption, 11
- symbolic execution, 14
- symbolic model, 9
 - for ZK, 28
- symbolically satisfy, 15
- symbolically-sound zero-knowledge, 28
- symmetric encryption
 - symbolic model for, 11
- term
 - (CryptoVerif), 22
 - ZK-, 27
- trace
 - computational node, 15
 - full, 15
 - full hybrid, 17
 - hybrid node, 18
 - node, 15
- trace property, 15
- true
 - ZK-atom, 27
 - ZK-proof, 27
- type
 - (CryptoVerif), 22
- unpredictability
 - of ZK proofs, 29
- unpredictable nonces, 12
- valid
 - ZK-atom, 27
- verification
 - deterministic, 29
- weak secrecy, 10
- zero-knowledge
 - proofs, 26
 - symbolic model, 28
 - symbolically-sound, 28
 - ZK, *see* zero-knowledge
 - ZK-circuit, 29
 - ZK-safe, 30
 - ZK-term, 27