A Secure Public-Key Authentication Scheme

Zvi Galil
Tel Aviv University
and
Columbia University
Department of Computer Science
New York, N.Y. 10027

Stuart Haber
Bellcore
445 South Street
Morristown, N.J. 07960

Moti Yung
IBM Research Division
T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, N.Y. 10598

Abstract

We propose interactive probabilistic public-key encryption schemes so that:

(1) the sender and the receiver of a message, as well as the message itself, can be authenticated;

(2) the scheme is secure against any feasible attack by a participant, including chosen-ciphertext attack.

Our suggested protocols can use any one-way trapdoor functions. In order to formulate and prove the properties of our procedures, we propose several new complexity-theoretic definitions of different levels of cryptographic security for systems which allow interaction. Chosen-ciphertext security is achieved using the techniques of minimum-knowledge interactive proofs, and requires only a constant number of message exchanges at the system's initiation stage.

J.J. Quisquater and J. Vandewalle (Eds.): Advances in Cryptology - EUROCRYPT '89, LNCS 434, pp. 3-15, 1990.

1 Introduction

To authenticate something is to prove that it is genuine or to establish its validity. There are several sorts of on-line authentication problem that face the designer of a cryptosystem. For example, he may wish to authenticate either a physical or a virtual channel between certain pairs of users; he may wish to allow users to establish their identities; or he may wish to enable the authentication of certain characteristics of specific messages.

Among the directions that researchers have taken in studying authentication problems are the "algebraic" approach of [14, 4, 5], the "authentication channel" of [15], the "digital signature" approach initiated by Diffie and Hellman [3] (which actually deals with a different problem that we do not study here), and the "identification scheme" of [6] (whose purpose is to identify a user without regard to specific messages).

In this paper we will assume that the communications media are not physically or otherwise authenticated. Our setting is that of the original Diffie-Hellman public-key model [3]: each user has an encryption key that is "public," or available to all other users (as well as to any adversary), and a corresponding private decryption key; all cryptographic tasks, including encryption and authentication of messages, must employ these keys. Our procedures are not based on the intractability of a specific computational problem, but are based instead on the more general assumption that a one-way trapdoor function exists.

In this setting, we distinguish several authentication problems that arise when A sends a message to B. There is the problem of *sender authentication*, which is to convince B that it was indeed A who sent the message, and the complementary problem of *receiver authentication*, which is to convince A that it is indeed B who received the message. There is also the problem of *message authentication*, which is to convince B that the message he receives is indeed the message that A intended to send.

In [7] we suggested an interactive procedure for the authentication both of the sender and of the receiver of a probabilistically encoded message. In this paper we extend our scheme so as to enable message authentication; and we show how to use interactive proofs of knowledge [16, 6] in order to refine our procedures so that they are secure against chosen-ciphertext attack. A major contribution of this paper is the formulation of definitions of security and privacy, using the language of computational complexity. In proving the properties of our constructions, we adapt the formal definitions of cryptographic security that were proposed by [17, 10], generalizing them to the setting of interacting users of a cryptosystem. Our definitions capture the intuitive notions that have often been used by cryptographers without careful definition, while allowing us to give precise proofs of the strength of the cryptographic procedures that we propose.

After describing our assumptions and notational conventions, we give our definitions of several different degrees of cryptographic security in §2. We give our authentication procedures and state their properties in §3. Finally, in §4, we formalize chosen-ciphertext

attack and give our procedures for achieving security against such attack. We give our new definitions in some detail in this abstract, but due to lack of space we omit the proofs in §§3-4; for these, see the full paper [9].

2 Definitions of cryptographic security

2.1 Preliminaries

We model the users of a cryptosystem as *interactive Turing machines*, as defined in [11, 8].

Our constructions assume the existence of a family of one-way functions with an associated family of hard-bit predicates, as follows [1, 17]. Let I be an *infinite* family of strings, and for each positive integer k let $I_k = \{i \in I \mid |i| = k\}$ denote the set of strings in I of length k. A *hard-bit family* is a family $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$, where for each i of length k, $f_i : D_i \to D_i$ is a function defined on a domain D_i consisting of k-bit strings, and $b_i : D_i \to \{0,1\}^{l(k)}$ is an $l(k)$-bit "predicate". We require that these sets be *accessible*: there is a probabilistic algorithm, running in expected polynomial time, that, given k, chooses $i \in I_k$ uniformly and at random; and there is another such algorithm that, given i, chooses $x \in D_i$ uniformly and at random. Both $(i, x) \mapsto f_i(x)$ and $(i, x) \mapsto b_i(x)$ are easy computations (i.e. of cost polynomial in k); the functions f_i are *one-way*, and furthermore it is computationally intractable to predict the bits of $b_i(x)$, given only the value $f_i(x) \in D_i$.

$$\begin{array}{ccc} x & \to & f_i(x) \\ \downarrow & \swarrow & \text{hard bits} \\ b_i(x) & & \end{array}$$

(Typically, one is given functions f_i that are assumed to be one-way, and then one proves that the "hard bits" $b_i \circ f_i^{-1}$ polynomially reduce to f_i^{-1}. By a recent result of Impagliazzo, Levin, and Luby [12], it suffices merely to assume the existence of a family $\{f_i\}$ of one-way functions.)

For our applications, we assume in addition that $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$ is a *trapdoor hard-bit family*: for each $i \in I$ there is a string d_i—the *trapdoor information* or the *secret key* associated with i—that enables the inversion of f_i, so that the computation $(i, y, d_i) \mapsto x \in f_i^{-1}(y)$ is easy, for any $y \in f_i(D_i)$. Furthermore, there is a probabilistic algorithm that, on input k, chooses a pair (i, d_i), with i uniformly distributed in the set I_k.

For the sake of greater efficiency in one of our constructions in §4 below, we assume that (as is the case with all the suggested examples of hard-bit families $\mathcal{F}$) the sets D_i are groups, and that each function f_i is a group automorphism.

The following presentation is adapted from that of [10, 13, 2]. We will say that a *public-key cryptosystem* consists of a *security parameter* k, a sequence of *message spaces*

$\{M_k\}$ (probability distributions on strings of polynomial length $l(k)$), a key-generator, possibly a public-key encryption-decryption algorithm pair, and one or more message-sending protocols:

- The *key-generator* is a probabilistic algorithm G that, on input 1^k, halts in expected time polynomial in k and writes out a pair of strings (e, d): a *public key* and a *secret key*. For example, for any trapdoor hard-bit family $\mathcal{F}$, there exists a corresponding key-generator $G_{\mathcal{F}}$ which, on input k, chooses $i \in I_k$ at random and writes out the pair (i, d_i).

- A *public-key encryption-decryption algorithm pair* consists of a pair of (possibly probabilistic) algorithms (E, D). The encryption algorithm E takes as input a message m chosen according to the message space M_k and a public key e with security parameter k, and produces a ciphertext c; when E is probabilistic, there may be many possible ciphertexts for each input pair (m, e). The decryption algorithm D takes as input a ciphertext c and a private key d, and produces the cleartext message m.

- A *message-sending protocol* is a pair of interacting probabilistic Turing machines (S, R), a *sender* and a *receiver*. Each machine takes as input a pair of public keys (e_S, e_R) (belonging to S and R, respectively) as well as its own private key d_S or d_R (respectively). The sender receives as additional input a message chosen according to the message space M_k. The machines proceed with their computation. At any point, either party may *reject* the execution, writing a special **reject** symbol on its private output tape. Otherwise, when they halt the receiver writes out, as a private output, the cleartext message m' that it has *accepted*. We also require that S write a message m (presumably, the one it has just tried to send) on its private output tape. The protocol is *correct* if $m' = m$ with overwhelming probability.

2.2 Security against cryptanalysis

In this section we repeat the definition first proposed by Goldwasser and Micali for the "polynomial security" of a probabilistic public-key encryption algorithm [10], and then we adapt the definition in order to handle more general public-key message-sending protocols. In our definitions, attacks are deemed to be successful if they result in very weak sorts of cryptanalytic ability; hence any procedure that satisfies one of our definitions is all the stronger against attacks that attempt to achieve more.

Suppose that we are given a public-key encryption-decryption algorithm pair (E, D) (along with a sequence of message spaces $\{M_k\}$ and a key-generator G). A *message-finder* is a family of probabilistic circuits $F = \{F_k\}$, as follows: on input a specific public key

with security parameter k, the circuit F_k produces as output two messages $m_0, m_1 \in M_k$. A *message-distinguisher* is a family of probabilistic circuits $D = \{D_k\}$, each of which takes four input strings and produces one output bit: the first input to D_k is a public key e with security parameter k; its second and third inputs are strings of length $l(k)$, for example a pair of messages chosen from M_k; and its fourth input is a string of length $l'(k)$, such as an encryption produced by E. (In [10] "message-distinguishers" were called "line-tappers.") Note that both F_k and D_k may have information about G, about M_k, and about E "hard-wired" into them.

Consider the following experiment. Run the algorithm G on input 1^k to produce a pair of keys (e, d); give the public key e as input to F_k to obtain a pair of messages $m_0, m_1 \in M_k$; choose one of these at random, m_b say, and give (m_b, e) to E to produce an encryption α; finally, give (e, m_0, m_1, α) as input to D_k to obtain the bit b'. (This bit b' may depend, of course, on the sequence of random bits used by these probabilistic circuits and probabilistic Turing machines.) The experiment is a success if $b = b'$. We say that the encryption-decryption algorithm pair (E, D) is *polynomially secure against cryptanalysis* if for any message-finder F, for any message-distinguisher D, and for any constant c, the probability of success is less than $\frac{1}{2} + \frac{1}{k^c}$ for sufficiently large k.

Suppose now that we are given a message-sending protocol (S, R) (along with $\{M_k\}$ and G). To analyze the security of such a protocol we define a *message-finder* to be a family of probabilistic circuits $F = \{F_k\}$ whose kth member takes as input a pair of public keys with security parameter k, and produces as output two messages $m_0, m_1 \in M_k$. Similarly, we define a *message-distinguisher* $D = \{D_k\}$ as before, except that D_k's inputs include a pair of public keys with security parameter k instead of just a single public key, and the transcript of a protocol execution instead of an output of the encryption algorithm. In addition, we give D_k another input string consisting of the random-bit string that was F_k's random input; this is a technical consideration that could conceivably increase the power of the attacking circuits. (Once again, both F_k and D_k may have information about M_k, about S, and about R "hard-wired" into them.)

As with a conventional public-key encryption algorithm, we consider the following experiment. Run the algorithm G twice on input 1^k to produce two pairs of keys (e_S, d_S) and (e_R, d_R), and give the public keys e_S, e_R (along with a random bit-sequence ρ) as input to F_k to obtain a pair of messages $m_0, m_1 \in M_k$. Choose one of these at random, m_b say, and use S and R to perform the given protocol to send the message m_b; let τ be the transcript of strings written by S and R on their communication tapes during the protocol execution. Finally, give $(e_S, e_R, \rho, m_0, m_1, \tau)$ as input to D_k to obtain the bit b'—its guess as to whether the message sent was m_0 or m_1. The experiment is a success if $b = b'$. We say that the message-sending protocol is *polynomially secure against cryptanalysis* if for any message-finder F, for any message-distinguisher D, and for any constant d, the probability of success is less than $\frac{1}{2} + \frac{1}{k^d}$ for sufficiently large k.

We remark that a conventional public-key encryption-decryption algorithm pair (E, D) can be regarded as a special kind of interactive message-sending protocol, one in which there is only one round of interaction. For such a message-sending protocol, the definition of security just proposed reduces to the Goldwasser-Micali definition described above.

Next we deal with the provision of authentication (for each message sent) in the context of the probabilistic public-key model. In §3 below we present our public-key solution to the problems of sender, receiver, and message authentication. In the rest of §2 we give our formal definitions of authentication security.

2.3 Security against sender impersonation

We formalize the property of sender authentication by asking what advantage an adversary can gain by impersonating the legitimate sender of a message. Thus we define a *sender-impersonator* to be an interactive Turing machine $\tilde{S}$ that takes as input a pair of public keys (e_S, e_R). Note that $\tilde{S}$ is not given either of the corresponding secret keys d_S, d_R.

Consider the following experiment. Run the algorithm G twice on input 1^k to produce two pairs of keys (e_S, d_S) and (e_R, d_R). Then perform the given protocol, with $\tilde{S}$ (instead of S) acting as the sender, and R performing the role of the receiver (as specified); let m be the private output of $\tilde{S}$. The experiment is successful if R accepts the same message m. We say that the protocol is *polynomially secure against sender impersonation* if for any sender-impersonator $\tilde{S}$ and for any constant d, the probability that the experiment is successful is less than $\frac{1}{2^{l(k)}} + \frac{1}{k^d}$ for sufficiently large k.

2.4 Security against receiver impersonation

We formalize the property of receiver authentication by asking what advantage an adversary can gain by impersonating the legitimate receiver of a message. Thus we define a *receiver-impersonator* to be an interactive Turing machine $\tilde{R}$ that takes as input a pair of public keys (e_S, e_R) and a pair of messages m_0, m_1. We define a *message-distinguisher* $D = \{D_k\}$ as above, except that the last input of D_k is $\tilde{R}$'s view of a protocol execution with security parameter k (in other words, a string of, say, $l'(k)$ bits). Note that D_k may have information about G, about M_k, about (S, R), and about $\tilde{R}$ "hard-wired" into it. (However, neither $\tilde{R}$ nor D_k is given either of the secret keys d_S, d_R corresponding to e_S, e_R.)

Let $F = \{F_k\}$ be a message-finder, and consider the following experiment. Run the algorithm G twice on input 1^k to produce two pairs of keys (e_S, d_S) and (e_R, d_R), and give the public keys e_S, e_R as input to F_k to obtain a pair of messages $m_0, m_1 \in M_k$. Choose one of these at random, m_b say, and perform the given protocol to send the message m_b, with S performing the role of the sender (as specified), and $\tilde{R}$ acting as the receiver. Finally,

give $\tilde{R}$'s view of the protocol execution to D_k to obtain a bit b' (its guess as to whether the message sent was m_0 or m_1). Call the experiment a success if $b = b'$. We say that the protocol is *polynomially secure against receiver impersonation* if for any message-finder F, for any receiver-impersonator $\tilde{R}$, for any message-distinguisher D, and for any constant d, the probability of success is less than $\frac{1}{2} + \frac{1}{k^d}$ for sufficiently large k.

2.5 Security against random-message attack

In order to address the problem of message authentication formally, we challenge an adversary, without knowing a sender's private key, to "force" a legitimate receiver to accept some legal message. Consider, therefore, the following experiment. Let $\tilde{S}$ be a sender-impersonator. Run the algorithm G twice on input 1^k to produce two pairs of keys (e_S, d_S) and (e_R, d_R). Then perform the given protocol, with $\tilde{S}$ (instead of S) acting as the sender, and R performing the role of the receiver (as specified). The experiment is successful if R accepts a message $m \in M_k$—any message at all. We say that the protocol is *polynomially secure against random-message attack* if for any sender-impersonator $\tilde{S}$ and for any constant d, the probability that the experiment is successful is less than $\frac{1}{k^d}$ for sufficiently large k.

Observe that a message-sending protocol which is polynomially secure against random-message attack is clearly also polynomially secure against sender impersonation.

3 Authentication schemes

Assume that we are given a trapdoor hard-bit family $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$, where b_i is an $l(k)$-bit predicate for $i \in I$ of length k. The following is a protocol for S to send an authenticated $l(k)$-bit message m to R. The sender's inputs include its public and secret keys (i_S, d_{i_S}), the receiver's public key i_R, and the message m; the receiver's inputs include its public and secret keys (i_R, d_{i_R}) and the sender's public key i_S. **Protocol 1**

1. $S \rightarrow R$: "Hi, this is S sending a message to R."
2. R chooses $x_S \in D_{i_S}$ at random, and computes $p_S := b_{i_S}(x_S)$ and $y_S := f_{i_S}(x_S)$.
   $R \rightarrow S$: y_S
3. S computes $x_S := f_{i_S}^{-1}(y_S)$ and $p_S := b_{i_S}(x_S)$, chooses $x_R \in D_{i_R}$ at random, and computes $p_R := b_{i_R}(x_R)$, $y_R := f_{i_R}(x_R)$, $c := m \oplus p_S \oplus p_R$.
   $S \rightarrow R$: $[c, y_R]$
4. R computes $x_R := f_{i_R}^{-1}(y_R)$, $p_R := b_{i_R}(x_R)$, $c \oplus p_S \oplus p_R = m$; and then R accepts and writes as output this message m.

end-protocol

Theorem 1 *Protocol 1, based on the trapdoor hard-bit family $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$, is a correct message-sending protocol that is polynomially secure against cryptanalysis, against sender impersonation, and against receiver impersonation.*

Protocol 1 is not secure against random-message attack. A cheating user S', not knowing S's trapdoor d_{i_S}, can still succeed in sending *some* message to R simply by choosing a string c at random in step 3. In order to foil such an attack, we adapt the protocol, as follows. In step 2, R chooses not just one but k random elements x_S, computing the corresponding values y_S and p_S for each one; similarly, S runs k versions of step 3 in parallel. Finally, R executes k versions of the computation $m' := c \oplus p_S \oplus p_R$, and only accepts a message if they all give the same value m'. Call the resulting procedure Protocol 2.

Theorem 2 *Protocol 2, based on the trapdoor hard-bit family $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$, is a correct message-sending protocol that has all the security properties of Protocol 1 and is also polynomially secure against random-message attack.*

4 Chosen-ciphertext security

The designer of a multi-user cryptosystem must be concerned not only with passive attacks or unauthenticated messages sent by impersonators but also with attacks carried on by active participants. One or more malicious users may take advantage of their user privileges, sending and receiving specially computed plaintext and ciphertext messages, or deviating from the system's message-sending protocols in some other manner, in order to attack a legitimate user's security. In this section we deal with such attacks.

4.1 Definition of chosen-ciphertext security

In a chosen-ciphertext attack on a standard encryption algorithm, the adversary is allowed to choose several ciphertext messages, and then is given the corresponding plaintext messages (if they exist). Generalizing to the context of message-sending protocols, we may regard such an attack as consisting of two stages: a participation stage during which the attackers take part in protocol executions and interact with other users, and an extraction stage during which the attackers try to infer additional information about legitimate users' messages or private keys. As in the definitions above, we will call the attack successful if the extraction stage results in a very weak sort of cryptanalytic ability.

As in §2.2 above, it may be helpful formally to define a chosen-ciphertext attack on a public-key encryption-decryption algorithm pair (E, D) before defining the attack on a more general message-sending protocol. A *chosen-ciphertext attack* on (E, D) consists of an interactive Turing machine $\mathcal{A}$, a message-finder $F = \{F_k\}$, and a message-distinguisher

$D = \{D_k\}$, as follows. The input to $\mathcal{A}$ consists of a security parameter k and a public key e; this key is the target of $\mathcal{A}$'s attack. Several times during the course of its computation, $\mathcal{A}$ interacts with D by requesting that a ciphertext string c of its choice be decrypted using the private key d that corresponds to e. (More precisely, $\mathcal{A}$ writes c on one of its communication tapes, and then reads $D(c,d)$ from its other communication tape.) The message-finder circuit F_k takes as input the target key e along with the contents of $\mathcal{A}$'s history tape, and computes as output two messages $m_0, m_1 \in M_k$. The message-distinguisher D_k takes as input a public key, the contents of an interactive Turing machine's history tape, a pair of messages in M_k, and an encryption produced by E.

Consider the following experiment. Run G on 1^k to get (e,d). Run $\mathcal{A}$ with input e, using D (and the private key d) to decrypt the requested strings; this computation is the participation stage of the attack. Next, in the extraction stage, give e and $\mathcal{A}$'s history tape h to F_k to obtain a pair of messages $m_0, m_1 \in M_k$. Choose one of these at random, m_b say, and give (m_b, e) to E to produce an encryption α; finally, give (e, h, m_0, m_1, α) as input to D_k to obtain the bit b'. The experiment is a success if $b = b'$. We say that (E,D) is *polynomially secure against chosen-ciphertext attack* if for any interactive Turing machine $\mathcal{A}$, for any message-finder F, for any message-distinguisher T, and for any constant d, the probability of success is less than $\frac{1}{2} + \frac{1}{k^d}$ for sufficiently large k.

Next we generalize this definition to deal with the more general setting of interactive protocols. Suppose that we are given a message-sending protocol (S,R) (along with $\{M_k\}$ and G). A *chosen-ciphertext attack* on the protocol consists of an interactive Turing machine $\mathcal{A}$, a message-finder $F = \{F_k\}$, and a message-distinguisher $D = \{D_k\}$, as follows. The input to $\mathcal{A}$ consists of a security parameter k and two public keys e_0 and e_1; these keys are the targets of $\mathcal{A}$'s attack. After performing several executions of the message-sending protocol, as described below, $\mathcal{A}$ writes out its view of these executions. The kth message-finding circuit F_k takes as input the target keys e_0, e_1 along with $\mathcal{A}$'s execution views, and computes as output two messages $m_0, m_1 \in M_k$ as well as a *choice* of either 0 or 1. The message-distinguisher is as above, except that D_k's inputs include a pair of public keys with security parameter k instead of a single public key, the machine $\mathcal{A}$'s history string, and the transcript of a protocol execution instead of an output of the encryption algorithm.

The attacking machine $\mathcal{A}$ is meant to model a coalition of several malicious users; it operates as follows. Several (polynomially many) times, $\mathcal{A}$ chooses to participate in the given message-sending protocol, each time choosing either to send or to receive a message, as well as choosing one of the two keys e_0 or e_1 to be used by the "legitimate" receiver or sender.

- If the choice is to send, then $\mathcal{A}$ takes the role of the sender in a protocol execution with R. In this execution, $\mathcal{A}$ may "send" a message from the message space M_k or a

message chosen according to some other computation. R uses either (e_0, d_0) or (e_1, d_1) according to $\mathcal{A}$'s choice. At the end of the execution, $\mathcal{A}$ is given R's private output (the message that R accepted). For convenience, we may refer to $\mathcal{A}$'s computation during this execution as a "subroutine" $\tilde{S}$.

- If the choice is to receive, then $\mathcal{A}$ takes the role of the receiver in a protocol execution with S, in which S attempts to send a message chosen according to the distribution M_k. S uses either (e_0, d_0) or (e_1, d_1) according to $\mathcal{A}$'s choice. At the end of the execution, $\mathcal{A}$ is given S's private output (the message that S tried to send). Once again, we may refer to $\mathcal{A}$'s "subroutine" $\tilde{R}$.

In each execution, $\mathcal{A}$ may request a new public-key, private-key pair generated by G (with security parameter k), or may use a pair requested earlier. These pairs are generated independently of the target keys.

$\mathcal{A}$'s attack may be "adaptive": any of its computation steps may depend on previous steps. For instance, the sending (or receiving) subroutine $\tilde{S}$ (or $\tilde{R}$) invoked by $\mathcal{A}$ during a particular execution of the protocol may be different from the subroutine used by $\mathcal{A}$ in an earlier execution. Of course, the computation of R (or of S)—modelling the actions of a legitimate receiver (or sender)—is not adaptive; each step is an independent execution with a new sequence of random coin-flips. Without loss of generality we may require that the concatenation of $\mathcal{A}$'s execution views be a string v of length at most $l(k)$, where l is a polynomial.

Consider the following experiment. Run G twice on input 1^k to obtain (e_0, d_0) and (e_1, d_1). Run $\mathcal{A}$ with input (e_0, e_1) as described above to produce the execution history v; this computation is the participation stage of the attack. Next, in the extraction stage, give (e_0, e_1) and v to F_k to produce $m_0, m_1 \in M_k$ and a "choice" of 0 or 1; this is a choice as to whether to use the target keys e_0 and e_1 as the receiver's key and the sender's key or vice versa in the next run of the protocol. Next, choose one of the messages m_0, m_1 at random, m_b say, and run (S, R) to send m_b, using as keys either (e_0, d_0) and (e_1, d_1) or (e_1, d_1) and (e_0, d_0) according to F_k's choice of 0 or 1, respectively; let τ be the ("public") transcript of this run. Finally, give the public keys e_0 and e_1, the pair of messages (m_0, m_1), the transcript τ, and the history string v as input to D_k to obtain the bit b'—its guess as to whether the message sent was m_0 or m_1. The experiment is a success if $b = b'$. We say that the message-sending protocol is *polynomially secure against chosen-ciphertext attack* if for any interactive Turing machine $\mathcal{A}$, for any message-finder F, for any message-distinguisher D, and for any constant d, the probability of success is less than $\frac{1}{2} + \frac{1}{k^d}$ for sufficiently large k.

4.2 Achieving chosen-ciphertext security

The main tool we use in order to achieve chosen-ciphertext security is the *zero-knowledge* (or *minimum-knowledge*) *interactive proof of knowledge* that was formalized by [16, 6]. In the form that we need it, this is a procedure whereby one party (the "prover") can prove to another party (the "verifier") that he "knows" or can compute a quantity without revealing to the verifier any computational knowledge about the value of that quantity. For example, if f_i is a one-way function and y is known beforehand to both parties, the prover can convince the verifier that he knows a pre-image of y, i.e. an element x that satisfies the relation $f_i(x) = y$, without revealing anything about the bits of x. In the case that each function f_i is a group automorphism of its domain D_i (thus providing an example of a "random self-reducible problem"), this interactive proof can be carried out especially efficiently.

We specify Protocol 3 by refining Protocol 1 in the following way. After sending a value y_S in step 2, R proves to S that he knows a preimage x_S. Similarly, after sending a value y_R in step 3, S proves to R that she knows a corresponding preimage x_R. At any point, if one of these interactive proofs is not successful—i.e. if the verifier is not "convinced"—then the unconvinced verifier rejects the attempted message-sending and halts the protocol.

The complete protocol is as follows. As in §3, this is a protocol for S to send an authenticated $l(k)$-bit message m to R. The sender's inputs include its public and secret keys (i_S, d_{i_S}), the receiver's public key i_R, and the message m; the receiver's inputs include its public and secret keys (i_R, d_{i_R}) and the sender's public key i_S.

Protocol 3

1. $S \to R$: "Hi, this is S sending a message to R."

2. R chooses $x_S \in D_{i_S}$ at random, and computes $p_S := b_{i_S}(x_S)$ and $y_S := f_{i_S}(x_S)$.
   $R \to S$: y_S

   R proves to S that it can compute an element of $f_{i_S}^{-1}(y_S)$; if the verifier would reject the proof, then S halts the protocol.

3. otherwise S computes $x_S := f_{i_S}^{-1}(y_S)$ and $p_S := b_{i_S}(x_S)$, chooses $x_R \in D_{i_R}$ at random, and computes $p_R := b_{i_R}(x_R)$, $y_R := f_{i_R}(x_R)$, $c := m \oplus p_S \oplus p_R$.
   $S \to R$: $[c, y_R]$

   S proves to R that it can compute an element of $f_{i_R}^{-1}(y_R)$; if the verifier would reject the proof, then R halts the protocol.

4. otherwise R computes $x_R := f_{i_R}^{-1}(y_R)$, $p_R := b_{i_R}(x_R)$, $c \oplus p_S \oplus p_R = m$; R accepts this message.

end-protocol

Theorem 3 *Protocol 3, based on the trapdoor hard-bit family $\mathcal{F} = \{(f_i, b_i) \mid i \in I\}$, is a correct message-sending protocol that has all the security properties of Protocol 1 and is also polynomially secure against chosen-ciphertext attack.*

We can refine Protocol 2 in a similar manner, so that the resulting protocol is polynomially secure against random-message attack and against chosen-ciphertext attack.

Finally, we briefly describe an adaptation of our protocols so as to provide a message-sending protocol that is polynomially secure against chosen-ciphertext attack, and requires overhead whose amortized cost (per bit of plaintext message sent) can be arbitrarily small.

In this protocol one of the two parties chooses a short random bit-string r, and then the two parties use the refined version of Protocol 2 so that he can send r to the other party. They then use r as a seed—known only to them—for a pseudo-random bit-generator so that they share a (simulated) one-time pad of length polynomial in the security parameter; different pieces of this one-time pad may be used in order to encode messages sent from either one of the two parties to the other. This enables the exchange of polynomially many messages, in such a way that the system as a whole is secure against chosen-ciphertext attack. This method minimizes the cryptographic tools required; it uses only the parties' public keys, with no additional keys produced in order to send additional messages.

References

[1] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comp.*, 13(4):850–864, Nov. 1984.

[2] G. Brassard. *Modern Cryptology: A Tutorial*, volume 325 of *Lecture Notes in Computer Science*. Springer-Verlag, 1988.

[3] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Trans. on Inform. Theory*, IT-22:644–654, November 1976.

[4] D. Dolev and A.C. Yao. On the security of public-key protocols. *IEEE Trans. on Inform. Theory*, IT-29:198–208, 1983.

[5] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th FOCS*, pages 34–39, 1983.

[6] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *J. of Cryptology*, 1(2):77–94, 1988.

[7] Z. Galil, S. Haber, and M. Yung. Symmetric public-key encryption. In *Crypto '85*. Springer-Verlag, 1986.

[8] Z. Galil, S. Haber, and M. Yung. Minimum-knowledge interactive proofs for decision problems. *SIAM J. Comput.*, 18(4):711–739, 1989.

[9] Z. Galil, S. Haber, and M. Yung. Symmetric public-key cryptosystems. Submitted for publication, 1989.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28:270–299, April 1984.

[11] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[12] R. Impagliazzo, L.A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proc. 21st STOC*, pages 12–24. ACM, 1989.

[13] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Comput.*, 17(2):412–426, 1988.

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

[15] G.J. Simmons. A survey of information authentication. *Proceedings of the IEEE*, 76(5):603–620, May 1988.

[16] M. Tompa and H. Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *Proc. 28th FOCS*, pages 472–482. IEEE, 1987.

[17] A.C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91. IEEE, 1982.