Maintaining Security in the Presence of Transient Faults *

Ran Canetti[†] Amir Herzberg[‡]

October 27, 1995

Abstract

Consider a multiparty system where parties may occasionally be "infected" by malicious, coordinated agents, called viruses. After some time the virus is expelled and the party wishes to regain its security. Since the leaving virus knows the entire contents of the infected party's memory, a source of "fresh" randomness seems essential for regaining security (e.g., for selecting new keys). However, such an "on-line" source of randomness may not be always readily available.

We describe a scheme which, using randomness only at the beginning of the computation, supplies each party with a new pseudorandom number at each round of communication. Each generated number is unpredictable by an adversary controlling the viruses, even if the party was infected in previous rounds. Our scheme is valid as long as in each round there is at least *one* noninfected party, and some of the communication links are secure.

We describe an important application of our scheme to secure sign-on protocols.

1 Introduction

Traditionally, cryptography was focused on protecting interacting parties (i.e., computers) against *external* malicious entities. Such cryptographic tasks include private communication over insecure channels, authentication of parties, unforgeable signatures, and general multiparty secure computation. An inherent property of all these scenarios is that once a party is "corrupted" it remains this way.

However, as computers become more complex, *internal* attacks on computers (i.e., attacks that corrupt components within a computer) have become an even more important security threat [LE93, Sto88]. Such attacks may be performed by internal (human) fraud, operating system weaknesses, or Trojan horse software (e.g. viruses). Security administrators often find internal attacks more alarming than external attacks, such as line tappings. An important property of internal attacks is that they are often temporary, or transient [ER89]. Thus the paradigm of "bad once means bad forever" does not hold here.

Still, almost no known solution to internal attacks allows recovery from faults. We briefly outline two main approaches underlying the known solutions to internal attacks. One approach to reducing the vulnerability of computers to internal attacks is to minimize the use of data that has

*Appears in the proceedings of Crypto 94.

[†]Department of Applied Mathematics and Computer Science, Weizmann Institute, Israel, e-mail: canetti@wisdom.weizmann.ac.il. Part of this research was done while visiting IBM T.J. Watson Research Center.
Supported by grant no. 92-00226 from the United States — Israel Binational Science Foundation, Jerusalem, Israel.

[‡]IBM T.J. Watson Research Center, POB 704, Yorktown Heights, NY 10598. E-mail: amir@watson.ibm.com.

to be kept secret. Most notable in this approach is the use of public key cryptosystems, where only the private key has to be kept secret (see, for instance, Novell NetWare 4.0). Still, the public key approach requires a secure, infallible certification entity. A different approach follows the popular paradigm of 'not putting all eggs in one basket'. That is, critical components are multiplied, and the overall security of the system is ensured as long as the attacker is unable to break all, or a large number, of the components.

However, in all these solutions, since the faults are assumed to be non-transient, there is no mechanism for taking advantage of a possible *recovery* of a component. This approach is contrasted with the traditional approach of fault-tolerance, which relies heavily on the fact that faults are transient, and on the reuse of recovered components. We believe that the idea of recovering and reusing components that have once been corrupted can be extremely useful also for cryptographic purposes. This idea, and in particular the recovery process, is the focus of this work.

Note that we may not know whether a particular component has been corrupted, and when the attackers have left; thus, the recovery process will be invoked periodically, regardless of whether we identified an attack. We call such an approach proactive, and say that such systems provide proactive security.

The goal of the recovery process is to ensure that once a component is recovered, it will again contribute to the overall security of the system. This goal is somewhat tricky. Even after the attacker loses control of a component, it still knows the (possibly modified) state of the component (e.g., the private cryptographic keys). Thus, a first step in the recovery process *must* be to somehow hand the recovering component some new secrets unknown to the attacker. These secrets can then be used to, say, choose new keys. The obvious way to generate such secrets is to use some source of "fresh", physical randomness. However, such a source may not be readily available or beneficial to use. In this paper we show that new secrets can be generated *without* fresh randomness. Instead, we use other, non-corrupted components in the system.

We consider a system (network) of components (parties) where every two parties are connected via a communication channel. (We elaborate below on the security requirements from the channels.) Parties may be temporarily corrupted (or infected) by malicious agents, called viruses. We assume that the viruses are controlled by an adversary. The adversary may choose to infect different parties at different times (i.e., communication rounds), as long as the *number* of infected parties is limited. We stress that there may be no party that has never been infected! This model was suggested by Ostrovsky and Yung [OY91], who also coined the terms 'infected' and 'viruses'.

We assume that, even if the faults are Byzantine, once the virus has left the party resumes executing its original *code* of the protocol (while the state may be corrupted). This assumption is explained as follows. If the virus can control the code after it leaves, then there is little meaning to recovery and it would be impossible to regain security. Furthermore, in practice there are reasonable ways to ensure that the code is not modified, such as physical read-only storage or comparison against backup copies. These techniques are used regularly in many systems.

In this work, we describe a scheme in which the parties use randomness *only at the beginning of the computation.* At each round, the scheme supplies each noninfected party with a "fresh" pseudorandom number, unpredictable by the adversary, even if this party was infected in previous rounds, and if the adversary knows all the other pseudorandom numbers supplied to any party at any round. In particular, these pseudorandom numbers can be used by a recovering party just as fresh random numbers (e.g., for regaining security). We call such a scheme a proactive pseudorandomness (PP) protocol.

Our implementation is simple, using pseudorandom functions [GGM86, GGM84]. We require the following very weak conditions. First, we assume that the adversary is computationally bounded. Next, we require that in each round of computation there is at least *one* secure party. A party is secure at a given round if it is noninfected at this round, and it has a secure (i.e., authenticated and private) link to a party that was secure in the previous round. We note that this link has to be secure only at this round.

1.1 Reconstructability and its Application to Secure Sign-On

1.1.1 Reconstructability.

Pseudorandom generators, being deterministic functions applied to a random seed, have the following advantage over truly random sources. A pseudorandom sequence is reconstructible, in the sense that it is possible to generate exactly the same sequence again by using the same seed. This property is very useful for several purposes, such as repeatable simulations and debugging, and is used in the construction of pseudo-random functions [GGM86]. Our application to secure sign-on protocols also makes use of this property.

In our setting, we say that a PP protocol is reconstructible if the value generated within each party at each round depends only on the seeds chosen by the parties at the beginning of the computation. In particular, these values should not depend on the random choices of the adversary.

Reconstructability is not easily achieved for proactive pseudorandomness protocols. In particular, the basic protocol described in this paper is reconstructible only if the faults are passive (namely, "eavesdropping" only). Crash faults (and also Byzantine faults, at the price of slightly compromising the security) could be tolerated by simple modifications described in section 5.

Application of Reconstructability to Secure Sign-On.

Unix and other operating systems provide security for the passwords by storing only a one-way function of the passwords on disk [MT79]. This technique allows authentication of the users, secure against eavesdropping the password file. Session security is not provided if the communication links are not secure.

In secure LAN systems, it is not realistic to assume that the communication is secure. Security mechanisms, therefore, avoid sending the password "on the clear". Instead, they use the user's password to derive a *session key*, with which they secure the communication. In both Kerberos [MNSS87] and NetSP / KryptoKnight[BGH+93a], this is done by using the password as a key for exchanging a random session key; this method also allows NetSP / KryptoKnight to authenticate the user automatically to additional systems ('single sign on').

However, this mechanism implies that some server must be able to compute the session key itself, using some secret (e.g. the password). This in turn implies that the server has to maintain the password file *secret*. This secrecy requirement is a major 'Achilles heel' of any security system. Indeed, NetWare 4.0 provides a more complicated and expensive solution, where the server keeps, for each user, an RSA private key encrypted using the user's password. The encrypted private key is sent to the workstation, which decrypts it using the password, and then uses it to derive a session key. This solution does not require the password file to be secret (but it is assumed that the password file is not modified).

We show how a reconstructible proactive pseudorandomness protocol can be used to overcome this weakness, without compromising efficiency. Our solution uses several proactive sign-on servers. The servers run a different copy of our PP protocol for each user. The initial seed of each server P_i is a pseudorandom value derived from the user's password, e.g. $f_{PW}(i)$. Each server sets its key for each time period to be the current output of the PP protocol. The user, knowing all the servers' inputs of this deterministic computation, can simulate the computation and compute each server's key at any time period *without need for any communication.* Thus, a user can always interact with the server of his choice. The security of our PP protocol makes sure that a mobile adversary does not know the key currently used by a secure server, as long as in each round there exists at least one secure server.

This solution does not require public key mechanisms, and is secure even if the attacker can modify the login files kept by the servers.

1.2 Related Works

Previous works have addressed the issue of transient faults in different ways. Reischuk [Rei85] designed a Byzantine agreement protocol which is able to tolerate faults covering a fraction of the network, if they remain stationary for a given interval of time. His results have been recently been extended by Garay [Gar94], who has also noted that randomization could provide a solution better in (expected) time by using the coin tossing techniques of [FM88]. Ostrovsky and Yung [OY91] showed how to perform secret sharing and how to compute any function in the presence of transient viruses. Their work required the parties to have a trusted random source, as well as complete security of the links; furthermore, their protocols tolerate only a small fraction of infected parties at each round. In [FY93], Franklin and Yung present a graph theoretic approach to privacy in a system with transient eavesdroppers.

We note that our PP protocol can be used to provide randomness to both [OY91] and [FM88].

Our application to single sign-on provides protection for both server and user against impersonation, even when attacker may break into servers. Traditional unix security [MT79] keeps only a hashing of the password, which allows an attacker (which reads the hashed-password file) to spoof the user. This has been extended by [BM93] to protect weak passwords from dictionary attack.

Organization. In Section 2 we define PP protocols, and recall the definition of pseudorandom function families. In Sections 3 and 4 we describe our PP protocol and prove its correctness. In Section 5 we describe some modifications to our application to secure sign-on protocols. In Appendix A we offer an alternative definition of PP protocols, and show that it is implied by our first definition.

2 Definitions

In this section we describe the communication model, the assumptions on the adversary, and define proactive pseudorandomness protocols.

Communication network. We consider a network where every two parties are connected via a secure (i.e., private and authenticated) communication channel. (In Section 4.1 we describe a relaxation of this security requirement on the links.) The parties are synchronized. Namely, the communication proceeds in rounds and all parties know the current round number. For simplicity, we also assume that at the end of each round, each party can send a message to each other party; these messages are received at the beginning of the next round. It is simple to extend our results to more realistic communication and synchronization models.

The Adversary. We assume that parties are occasionally being infected by some external, malicious entities, called viruses. Upon infecting a party, the entire contents of the party's memory becomes known to the virus. Furthermore, the virus can alter the party's memory and program. After some time the virus is discovered and removed from the party. Once a virus is removed, the party returns to execute its original program; however, its memory may have been altered.

Assuming that the rate in which new parties are infected is roughly equal to the rate in which the viruses are being removed, we may model this scenario as follows. We assume a limit, t, on the number of viruses. The adversary may decide which parties are infected at each round, as long as at each round the number of infected parties is at most t.

We assume that the viruses cooperate. That is, there exists an entity that accumulates all the data gathered by the viruses and coordinates their activity. We call this entity a mobile adversary. We say that a mobile adversary is t-limited if in each round of the computation at most t parties are infected. (We stress that there may exist no party that has never been infected!)

A definition of proactive pseudorandomness. Consider a network of parties performing some computation in the presence of a mobile adversary. We limit the parties' access to randomness to the beginning of the computation. Once the interaction starts no additional randomness is available. Still, the parties may need secret, "fresh" random input at different rounds of the computation (e.g., in order to recover from a virus).

For this purpose, the parties will run a special deterministic protocol; this protocol will generate a new value within each party at each round. Given that the parties' initial inputs of this protocol are randomly chosen, the value generated within each party at each round will be indistinguishable from random from the point of view of a mobile adversary, *even if the values generated within all the other parties, at all rounds, are known.* We call such a protocol a proactive pseudorandomness (PP) protocol.

We stress that at each round the adversary may know, in addition to the data gathered by the viruses, the outputs of all the parties (including the noninfected ones) at all the previous rounds. Still, it cannot distinguish between the current output of a noninfected party and a random value.

More specifically, consider the following attack, called an on-line attack, with respect to an n-party PP protocol. Let the input of each party be taken at random from $\{0,1\}^k$, where k is a security parameter (assume $n < k$). Furthermore, each party's output at each round is also a value in $\{0,1\}^k$.

On-line attack: *The protocol is run in the presence of a mobile adversary for m rounds (m is polynomial in n and k), where in addition to the data gathered by the viruses, the adversary knows the outputs of all the parties at all the rounds. At a certain round, l (chosen "on-line" by the adversary), the adversary chooses a party, P, out of the noninfected parties at this round. The adversary is then given a test value, v, instead of P's output at this round. The execution of the protocol is then resumed for rounds $l+1, \ldots, m$.* (Our definition will require that the adversary be unable to say whether v is P's output at round l, or a random value.)

For an n-party protocol π, and a mobile adversary A, let $A(\pi, \mathrm{PR})$ (respectively, $(A(\pi, \mathrm{R}))$ denote the output of A after an on-line attack on π, and when the test value v given to A is indeed the output of the specified party (respectively, when v is a random value). Without loss of generality, we assume that $A(\pi, \mathrm{PR}) \in \{0,1\}$.

Definition 1 *Let π be a deterministic n-party protocol with security parameter k. We say that π is a t-resilient* proactive pseudorandomness protocol (PP) *if for every t-limited polynomial time mobile adversary A, for all $c > 1$ and all large enough k we have*

$$|\mathrm{Prob}(A(\pi, \mathrm{PR}) = 1) - \mathrm{Prob}(A(\pi, \mathrm{R}) = 1)| \leq \frac{m}{k^c}$$

where m is the total number of rounds of protocol π, and the probability is taken over the parties' inputs of π and the choices of A. (We stress that m is polynomial in k.)

We say that π is efficient *if it uses resources polynomial in n and k.*

An alternative definition. Using Definition 1 above, it can be shown that the following property holds for any randomized application protocol α run by the parties. Consider a variant, α', of α that runs a PP protocol π along with α, and uses the output of π as random input for α at each round. Then, The parties' outputs of α and α' are indistinguishable; furthermore, running α' the adversary gains no knowledge it did not gain running α. In fact, this property may serve as an alternative definition for PP protocols. In Appendix A we present a more precise definition of this property.

We can now state our main result:

Theorem 1 *If one-way functions exist, then for all $n \in \mathbf{N}$ there exists an efficient, $(n-1)$-resilient PP protocol for n parties.*

Pseudorandom function families. Our constructions make use of pseudorandom functions families. We briefly sketch the standard definition.

Let $\mathcal{F}_k$ denote the set of functions from $\{0,1\}^k$ to $\{0,1\}^k$. Say that an algorithm D with oracle access distinguishes between two random variables f and g over $\mathcal{F}_k$ with gap $s(k)$, if the probability that D outputs 1 with oracle to f differs by $s(k)$ from the probability that D outputs 1 with oracle to g. Say that a random variable f over $\mathcal{F}_k$ is $s(k)$-pseudorandom if no polynomial time (in k) algorithm with oracle access distinguishes between f and $g \in_R \mathcal{F}_k$ with gap $s(k)$. (Throughout the paper, we let $e \in_R D$ denote the process of choosing an element e uniformly at random from domain D.)

We say that a function family $F_k = \{f_\kappa\}_{\kappa \in \{0,1\}^k}$ (where each $f_\kappa \in \mathcal{F}_k$) is $s(k)$-pseudorandom if the random variable f_κ where $\kappa \in_R \{0,1\}^k$ is $s(k)$-pseudorandom. A collection $\{F_k\}_{k \in \mathbf{N}}$ is pseudorandom if for all $c > 0$ and for all large enough k, the family F_k is $\frac{1}{k^c}$-pseudorandom. We consider pseudorandom collections which are efficiently constructible. Namely, there exists a polytime algorithm that on input $\kappa, x \in \{0,1\}^k$ outputs $f_\kappa(x)$.

Pseudorandom function families and their cryptographic applications were introduced by Goldreich, Goldwasser and Micali [GGM86, GGM84]. Applications to practical key distribution and authentication protocols were shown by Bellare and Rogaway [BR93]. In [GGM86] it is shown how to construct pseudo-random functions from any pseudo-random generator, which in turn could be constructed from any one-way function [HILL93]. However, practitioners often trust and use much simpler constructions based on DES or other widely available cryptographic functions.

3 The Protocol

In this section we describe the basic protocol. Several modifications useful for the application to secure sign-on are described in Section 5.

Consider a network of n parties, $P_1, \ldots, P_n$, having inputs $x_1, \ldots, x_n$ respectively. Each input value x_i is uniformly distributed in $\{0,1\}^k$, where k is a security parameter. We assume that parties have agreed on a predefined pseudorandom function family $F = \{f_\kappa\}_{\kappa \in \{0,1\}^k}$, where each $f_\kappa : \{0,1\}^k \to \{0,1\}^k$.

In each round l each party P_i computes an internal value (called a key), $\kappa_{i,l}$, in a way described below. P_i's output at round l, denoted $r_{i,l}$, is set to be $r_{i,l} = f_{\kappa_{i,l}}(0)$, where 0 is an arbitrary fixed value.

The key $\kappa_{i,l}$ is computed as follows. Initially, P_i sets its key to be its input value, namely $\kappa_{i,0} = x_i$. At the end of each round $l \geq 0$, party P_i sends $f_{\kappa_{i,l}}(j)$ to each party P_j. Next, P_i *erases* its key for round l and sets its key for round $l+1$ to the bitwise exclusive or of the values received from all the parties at this round:

$$\kappa_{i,l+1} = \oplus_{j=1}^{n} f_{\kappa_{j,l}}(i) \tag{1}$$

We stress that it is crucial that the parties *erase* the old keys. In fact, if parties *cannot erase* their memory, proactive pseudo-randomness is impossible. In particular, once each party has been infected in the past, the adversary has complete information on the system at, say, the first round. Now the adversary can predict all the subsequent outputs of this deterministic protocol.

4 Analysis

We first offer some intuition for the security of our protocol. This intuition is based on an inductive argument. Assume that, at round l, the key of a noninfected party is pseudorandom from the point of view of the adversary. Therefore, the value that this party sends to each other party is also pseudorandom. Furthermore, the values received by different parties seem unrelated to the adversary; thus, the value that each party receives from a noninfected party is pseudorandom from the point of view of the adversary, even if the values sent to other parties are known. Thus, the value computed by each noninfected party at round $l+1$ (being the bitwise exclusive or of the values received from all the parties) is also pseudorandom.

Naturally, this argument serves only as intuition. The main inaccuracy in it is in the implied assumption that we do not lose any pseudo-randomness in the repeated applications of pseudorandom functions. A more rigorous proof of correctness (using known techniques) is presented below.

Theorem 2 *Our protocol, given a pseudorandom function family, is an efficient, $(n-1)$-resilient PP protocol.*

Proof: Let π denote our protocol (run for m rounds). Assume there exists a polytime mobile adversary A such that

$$|\text{Prob}(A(\pi, \text{PR}) = 1) - \text{Prob}(A(\pi, \text{R}) = 1)| > \frac{m}{k^c}$$

for some constant $c > 0$ and some value of k. For simplicity we assume that A infects exactly $n-1$ parties at each round, and that A always runs the full m rounds before outputting its guess. The proof can be easily generalized to all A. We show that F_k is not pseudorandom. Specifically, we construct a distinguisher D that distinguishes with gap $\frac{1}{2k^c}$ between the case where its oracle is taken at random from F_k and the case where its oracle is a random function in $\mathcal{F}_k$.

In order to describe the operation of D, we define hybrid probabilities as follows. First, define $m+1$ hybrid protocols, $H_0, \ldots, H_m$, related to protocol π. Protocol H_i instructs each party P_s to proceed as follows.

- In rounds $l \le i$ party P_s outputs a random value and sends a random value to each other party P_t (instead of $f_{\kappa_{s,l}}(0)$ and $f_{\kappa_{s,l}}(t)$, respectively). In other words, P_s uses a random function from $\mathcal{F}_k$ instead of $f_{\kappa_{s,l}}$ for his computations.

- In rounds $l > i$ party P_s executes the original protocol, π.

Ofcourse, whenever a party is infected it follows the instructions of the adversary.

Distinguisher D, given oracle access to function g, operates as follows. First, D chooses at random a round number $l_0 \in_R [0, \ldots, m-1]$. Next, D runs adversary A on the following simulated on-line attack on a network of n parties. The infected parties follow the instructions of A. The (single) noninfected party at each round l, denoted $P_{*(l)}$, proceeds as follows.

1. In rounds $l < l_0$, party $P_{*(l)}$ outputs a random value and sends a random value to each other party (as in the first steps of the hybrid interactions).

2. In round l_0, party $P_{*(l_0)}$ uses the oracle function g to compute its output and messages. Namely, it outputs $g(0)$ and sends $g(j)$ to each other party P_j.

3. In rounds $l > l_0$, party $P_{*(l)}$ follows protocol π.

(Note that D knows which parties are infected by A at each round.)

Once a round is completed, D reveals all the parties' outputs of this round to A (as expected by A in an on-line attack). When A asks for a test value v, D proceeds as follows. First, D chooses a bit $b \in_R \{0,1\}$. If $b = 0$, then D sets v to the actual corresponding output of the party chosen by A. Otherwise, D sets v to a random value. Finally, if $b = 0$ then at the end of the simulated interaction D outputs whatever A outputs. If $b = 1$ then D outputs the *opposite value* to whatever A outputs.

The operation of D can be intuitively explained as follows. It follows from a standard hybrids argument that there must exist an i such that either $|\text{Prob}(A(H_i, \text{PR}) = 1) - \text{Prob}(A(H_{i+1}, \text{PR}) = 1)|$ is large or $|\text{Prob}(A(H_i, \text{R}) = 1) - \text{Prob}(A(H_{i+1}, \text{R}) = 1)|$ is large. Thus, if D chooses the "correct" values for l_0 and b it can use the output of A to distinguish between the two possible distributions of its oracle. We show that a similar distinction can be achieved if l_0 and b are chosen at random.

We analyze the output of D as follows. Let $\text{PR}_i \triangleq \text{Prob}(A(H_i, \text{PR}) = 1)$. (Namely, PR_i is the probability that A outputs 1 after interacting with parties running protocol H_i and when the test value given to A is indeed the corresponding output value of the party that A chose.) Similarly, let $\text{R}_i \triangleq \text{Prob}(A(H_i, \text{R}) = 1)$. Let ρ (resp., ϕ) be a random variable distributed uniformly over $\mathcal{F}_k$ (resp., over F_k). Assume that D is given oracle access to ρ. Then, at round l_0 party $P_{*(l_0)}$ outputs a random value and sends random values to all the other parties. Thus, the simulated interaction of A is in fact an on-line attack of A on protocol H_{l_0}. Therefore, if $b = 0$ (resp., if $b = 1$), then D outputs 1 with probability PR_{l_0} (resp., $1 - \text{R}_{l_0}$). Similarly, D is given oracle access to ϕ then the simulated interaction of A is an on-line attack of A on protocol H_{l_0+1}. In this case, if $b = 0$ (resp., if $b = 1$), then D outputs 1 with probability PR_{l_0+1} (resp., $1 - \text{R}_{l_0+1}$). Thus,

$$\begin{aligned}
\text{Prob}(D^{\rho} = 1) - \text{Prob}(D^{\phi} = 1) &= \frac{1}{2m}\sum_{i=0}^{m-1}(\text{PR}_i + 1 - \text{R}_i) - \frac{1}{2m}\sum_{i=0}^{m-1}(\text{PR}_{i+1} + 1 - \text{R}_{i+1}) \\
&= \frac{1}{2m}\sum_{i=0}^{m-1}[(\text{PR}_i - \text{R}_i) - (\text{PR}_{i+1} - \text{R}_{i+1})] \\
&= \frac{1}{2m}[(\text{PR}_0 - \text{R}_0) - (\text{PR}_m - \text{R}_m)].
\end{aligned}$$

Clearly, H_0 is the original protocol π. Thus, by the contradiction hypothesis, $|\text{PR}_0 - \text{R}_0| > \frac{m}{k^c}$. On the other end, in protocol H_m the parties output random values in all the m rounds, thus $\text{PR}_m - \text{R}_m = 0$. We conclude that $|\text{Prob}(D^{\rho} = 1) - \text{Prob}(D^{\phi} = 1)| > \frac{1}{2m} \cdot \frac{m}{k^c} = \frac{1}{2k^c}$. □ □

4.1 Insecure Links

When describing the model, we assumed that all the communication links are secure (i.e., private and authenticated). Here, we discuss the effect of insecure links on our protocol. We note that the protocol remains a PP protocol even if in each round l only a single noninfected party P_i has a single link which is secure in round l to a party P_j that wasn't infected in round $l-1$ (and P_j had in round $l-1$ a secure link to a noninfected party, etc.).

This security requirement on the links is minimal in the following sense. If no randomness is allowed after the interaction begins then a mobile adversary that sees the entire communication can continue simulating each party that has once been infected, even after the virus has left this party. Thus, after few rounds, the adversary will be able to simulate all parties and predict the output of each party at each subsequent round.

5 On the Application to Secure Sign-On

In subsection 1.1 we discussed reconstructible protocols and described an application of our PP protocol to proactive secure sign-on, using its reconstructability. However, as mentioned there, the protocol described in Section 3 is reconstructible only if all the parties (servers) follow their protocols at all times (that is, the adversary is only eavesdropping).

In this section we describe modifications of our protocol, aimed at two goals: one goal is to make the protocol more efficient for the user; the other goal is to maintain the reconstructability property for the case where the servers don't follow their protocols.

We start by describing a variant of the protocol which is more efficient for the user. Using the protocol described in Section 3, the user had to simulate the computation performed by the servers step by step. In case that many rounds have passed since the last time the user updated its keys, this may pose a considerable overhead. Using this variant, denoted ρ, the user can compute its updated key simulating only *one* round of computation of the servers. On the other hand, this variant has a weaker resilience property: it assures that the servers' keys be unpredictable by the adversary only if there exists a server that has never been infected.

The variant is similar in structure to the original protocol with the following modification. Each P_i has a master key which is never erased. This master key is set to be the initial key, $\kappa_{i,0}$ (derived, say, from the password). The master key is used as the index for the function at all the rounds. Namely, the key $\kappa_{i,l}$ at round l is computed as follows:

$$\kappa_{i,l} = \oplus_{j=1}^{n} f_{\kappa_{j,0}}(i)$$

It is also possible to combine the original protocol with the variant described above, in order to reach a compromise between efficiency and security. We define a special type of round: a major round. (For instance, let every 10th round be a major round.) The parties now update their master keys, using the original protocol, only at major rounds. In non-major rounds, the servers use their current master key as the index for the function.

This combined protocol has the following properties. On one hand, the user has one tenth of the rounds to simulate than in the original scheme. On the other hand, we only need that in any period of 10 rounds there exists a server that has not been infected. We believe that such versions may be a more reasonable design for actual implementations.

Next, we describe additions to the protocol, aimed at maintaining the reconstructability property for the case where the servers don't follow their protocols. We note that it is possible to withstand crash failures of servers, if the servers cooperatively keep track of which servers crashed at each round (this coordination can be done using standard consensus protocols).

The following addition to the protocol handles the case of Byzantine faults, if variant ρ described above is used. The only way an adversary controlling party P_j could interfere with the reconstructability of variant ρ is by sending a wrong value instead of $f_{\kappa_{j,0}}(i)$, to some server P_i.

However, P_i can, when unable to authenticate a user, compare each of the $f_{\kappa_{j,0}}(i)$ values with the user, e.g. using the 2PP protocol [BGH+93b], without exposing any of the values. If there are more than half of the values which match, the server and the user may use the exclusive or of these values only. This technique requires that at any round, the majority of the servers are non-faulty (otherwise the server may end up using values which are all known to the adversary). We note that this idea does not work if the basic protocol (that of Section 3) is used instead of variant ρ.

Conclusions and Open Problems

We believe that incorporating proactive security (i.e., the repeated, periodic attempt at recovery from potential break-ins, using other components in the system) in the design of systems can greatly inhance their security. In particular, our proactive pseudorandomness protocol may prove very useful in the design of proactive security protocols, supplying them with pseudorandomness (with some reconstructibility properties).

We propose the following open problem as a particularly challenging example for the applicability of proactive security to many security tasks. One of the major drawbacks of identity based cryptosystems is their reliance on a *single, trusted* key-generation facility, which knows the keys of all parties. It would be much better if one could design a distributed key-generation facility composed of several, potentially less trusted servers, but having proactive security.

Another challenging and important problem is to find a PP protocol with full reconstructibility against byzantine faults without giving up on security (preliminary results have been obtained by [CH94]).

Acknowledgments

Many thanks to Oded Goldreich and Hugo Krawczyk, who helped us with many constructive comments and suggestions. Thanks also to Chee-Seng Chow, Juan Garay and Moti Yung, for their support and comments.

References

[BGH+93a] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. A family of light-weight protocols for authentication and key distribution. Submitted to IEEE T. Networking, 1993.

[BGH+93b] Ray Bird, Inder Gopal, Amir Herzberg, Phil Janson, Shay Kutten, Refik Molva, and Moti Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, June 1993. Special issue on Secure Communications. See also a different version in *Crypto 91*.

[BM93] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: a password based protocol secure against dictionary attacks and password file compromise. In *1st ACM Conference on Computer and Communications Security*, pages 244–250, November 1993.

[BR93] Mihir Bellare and Phil Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: Proc. of Crypto 93*, pages 232–249, August 1993.

[CH94] Chee-Seng Chow and Amir Herzberg. A reconstructible proactive pseudo-randomness protocol. Work in progress, June 1994.

[ER89] M. Elchin and J. Rochlis. With microscope and tweezers: An analysis of the internet virus of november 1988. In *IEEE Symp. on Security and Privacy*, pages 326–343, 1989.

[FM88] P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the* 20^{th} *Annual ACM Symposium on Theory of Computing*, pages 148–161, May 1988.

[FY93] Mat Franklin and Moti Yung. Eavesdropping games: A graph-theoretic approach to privacy in distributed systems. In 34^{th} *Annual Symposium on Foundations of Computer Science*, pages 670–679, November 1993.

[Gar94] Juan A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults. To be presented in WDAG, 1994.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. On the cryptographic applications of random functions. In *Advances in Cryptology: Proc. of Crypto 84*, pages 276–288, 1984.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. Extended abstract in FOCS84.

[HILL93] John Hästad, Russell Impagliazzo, Leonid Levin, and Mike Luby. Construction of pseudo-random generator from any one-way functions. Manuscript, see preliminary versions by Impagliazzo et al. in *21st STOC* and Hästad in *22nd STOC*, 1993.

[LE93] Thomas A. Longstaff and Schultz E. Eugene. Beyond preliminary analysis of the wank and oilz worms: A case of study of malicious code. *Computers and Security*, 12(1):61–77, 1993.

[MNSS87] S. P. Miller, C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. In *Project Athena Technical Plan.* Massachusetts Institute of Technology, July 1987.

[MT79] R. H. Morris and K. Thompson. Unix password security. *Comm. ACM*, 22(11):594–597, November 1979.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the* 10^{th} *Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada*, pages 51–59, 1991.

[Rei85] R. Reischuk. A new solution to the byzantine generals problem. *Information and Control*, pages 23–42, 1985.

[Sto88] Cliff Stoll. How secure are computers in the u.s.a.? *Computers and Security*, 7(6):543–547, 1988.

Appendix A: An alternative definition of PP

We offer an alternative definition of a PP. This definition follows from Definition 1. Borrowing from the theory of secure encryption functions, we call Definition 1 "PP in the sense of indistinguishability" (or, in short, PP), whereas Definition 3 below is called "semantic PP" (or, in short, SPP). We first recall the standard definition of polynomial indistinguishability of distributions.

Definition 2 *Let $\mathcal{A} = \{A_k\}_{k\in\mathbf{N}}$ and $\mathcal{B} = \{B_k\}_{k\in\mathbf{N}}$ be two ensembles of probability distributions. We say that $\mathcal{A}$ and $\mathcal{B}$ are* polynomially indistinguishable *if there exists a constant $c > 0$ such that for every polytime distinguisher D and for all large enough k,*

$$|\mathrm{Prob}(D(A_k) = 1) - \mathrm{Prob}(D(B_k) = 1)| < \frac{1}{k^c}.$$

We colloquially let $A_k \approx B_k$ denote "$\{A_k\}$ and $\{B_k\}$ are polynomially indistinguishable".

Let α be some distributed randomized protocol, which is resilient against t-limited mobile adversaries, for some value of t. We wish to adapt protocol α to a situation where no randomness is available once the interaction starts. Namely, we want to construct a protocol α' in which the parties use randomness only before the interaction starts, and the parties' outputs of protocol α' are "the same as" their outputs of protocol α.

A general framework for a solution to this problem proceeds as follows. The parties run protocol α along with another *deterministic* protocol, π. Each party's local input of protocol π is chosen at random at the beginning of the interaction. In each round, each party sets the random input of protocol α for this round to be the current output of protocol π. We call π a semantically secure proactive pseudorandomness(SPP) protocol. We refer to protocol α as the application protocol.

We state the requirements from a SPP protocol π. Informally, we want the following requirement to be satisfied for every application α. Whatever an adversary can achieve by interacting with α combined with protocol π as described above, could also be achieved by interacting with the original protocol α when combined with a truly random oracle. More formally,

- for an n-party randomized application protocol α, a mobile adversary A, an input vector $\vec{x} = x_1, \ldots, x_n$, and $1 \leq i \leq n$, let $\alpha(\vec{x}, A)_i$ denote party P_i's output of protocol α with a random oracle when party P_j has input x_j and in the presence of adversary A. Let $\alpha(\vec{x}, A)_0$ denote A's output of this execution. Let $\vec{\alpha}(\vec{x}, A) \triangleq \alpha(\vec{x}, A)_0, \ldots, \alpha(\vec{x}, A)_n$.

- For a randomized protocol α and a deterministic protocol π for which each party has an output at each round, let $\alpha\pi$ denote the protocol in which α and π are run simultaneously and each party at each round sets the random input of α to be the current output of π.

Definition 3 *We say that an n party deterministic protocol π is a t-resilient* SPP *protocol if for every (randomized) application protocol α and every t-limited mobile adversary A there exists a t-limited mobile adversary A' such that for every input vector $\vec{x}$ (for protocol α) we have*

$$\vec{\alpha\pi}(\vec{x}, A') \approx \vec{\alpha}(\vec{x}, A).$$

where the probabilities are taken over the inputs of π and the random choices of A and of the oracle of α.

Theorem 3 *If a protocol is a t-resilient PP protocol then it is a t-resilient SPP protocol.*