Chapter 6

Proactive security

We introduce a method for maintaining the security of systems in the presence of repeated, however transient break-ins to system components. We use secure multiparty computation as a formal setting, and use mobile adversaries as a vehicle for concentrating on the mechanism of recovery from break-ins. This construction has applications to key management schemes in actual security systems. See Section 1.6 for an introductory presentation.

In Section 6.1 we define PP protocols, and recall the definition of pseudorandom function families. In Sections 6.2 and 6.3 we describe our PP protocol and prove its correctness. In Section 6.4 we describe some modifications to our application to secure sign-on protocols. In Section 6.5 we offer an alternative definition of PP protocols, and show that it is implied by our first definition.

6.1 Definitions

The setting. We consider a synchronous network with secure channels, and computationally bounded mobile adversaries. (In Section 6.3.1 we describe a relaxation of this security requirement on the channels.) For simplicity, we 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. (We remark that here the rounds formalize the applications of the automatic recovery mechanism described in Section 1.6. These are different than communication rounds. Typically, a recovery round may take place every several days, where a communication round lasts only a fraction of a second.)

The Adversary. At the beginning of each round the adversary may corrupt parties. (The adversary adaptively decides which parties to corrupt at each round.) Upon corruption of a party, the entire contents of the party's memory becomes known to the adversary. Furthermore, the adversary can alter the party's memory and program. After some time the adversary leaves the party. Once the adversary has left, the party returns to execute its original program; however, its memory may have been altered. We call this adversary a mobile adversary. We say that a mobile adversary is t-limited if in each round at most

t parties are corrupted. (We stress that there may exist no party that has never been corrupted!)

A definition of proactive pseudorandomness. Consider a network of parties performing some computation in the presence of a mobile adversary. The parties have access to randomness only at the beginning of the computation. Once the interaction starts no additional randomness is available.

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 adversary, the outputs of all the parties (including the uncorrupted ones) at all the previous rounds. Still, it cannot distinguish between the current output of an uncorrupted party and a random value.

More formally, 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 adversary, 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 uncorrupted 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.)

F or 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 6.1 *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 6.1 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 Section 6.5 we present a more precise definition of this property.

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 $g \in_{\mathrm{R}} \mathcal{F}_k$ with gap $s(k)$. (Throughout the paper, we let $e \in_{\mathrm{R}} D$ 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 $\kappa \in_{\mathrm{R}} \{0,1\}^k$ 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 [GGM2, GGM1]. Applications to practical key distribution and authentication protocols were shown by Bellare and Rogaway [BR1]. In [GGM2] 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 [HILL]. However, practitioners often trust and use much simpler constructions based on DES or other widely available cryptographic functions.

6.2 The Protocol

In this section we describe the basic protocol. Several modifications useful for the application to secure sign-on are described in Section 6.4.

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 $f_\kappa : \{0,1\}^k \rightarrow \{0,1\}^k$.

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:

$$\kappa_{i,l+1} = \oplus_{j=1}^{n} f_{\kappa_{j,l}}(i) \tag{6.1}$$

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 corrupted 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.

6.3 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 an uncorrupted 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 an uncorrupted 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 uncorrupted 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 6.2 *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 corrupts 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 $l \leq i$ 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 corrupted 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 $l_0 \in_{\mathrm{R}} [0, \ldots, m-1]$. Next, D runs adversary A on the following simulated on-line attack on a network of n parties. The corrupted parties follow the instructions of A. The (single) uncorrupted 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 corrupted 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 $b \in_{\mathrm{R}} \{0, 1\}$. 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 $|\mathrm{Prob}(A(H_i, \mathrm{PR}) = 1) - \mathrm{Prob}(A(H_{i+1}, \mathrm{PR}) = 1)|$ is large or $|\mathrm{Prob}(A(H_i, \mathrm{R}) = 1) - \mathrm{Prob}(A(H_{i+1}, \mathrm{R}) = 1)|$ 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 $\mathrm{PR}_i \triangleq \mathrm{Prob}(A(H_i, \mathrm{PR}) = 1)$. (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 $\mathrm{R}_i \triangleq \mathrm{Prob}(A(H_i, \mathrm{R}) = 1)$. 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., $1 - \mathrm{R}_{l_0}$). 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., $1 - \mathrm{R}_{l_0+1}$). Thus,

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

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

6.3.1 Insecure Links

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

This security requirement on the channels 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 corrupted, even after the adversary 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.

6.4 On the Application to Secure Sign-On

In subsection 1.6.1 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 6.2 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 6.2, 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 corrupted.

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

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 corrupted. 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+2], 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 6.2) is used instead of variant ρ.

6.5 An alternative definition of PP

We offer an alternative definition of a PP. This definition follows from Definition 6.1. Borrowing from the theory of secure encryption functions, we call Definition 6.1 "PP in the sense of indistinguishability" (or, in short, PP), whereas Definition 6.4 below is called "semantic PP" (or, in short, SPP). We first recall the standard definition of polynomial indistinguishability of distributions.

Definition 6.3 *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,*

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

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 $\vec{\alpha}(\vec{x}, A) \stackrel{\triangle}{=} \alpha(\vec{x}, A)_0, \ldots, \alpha(\vec{x}, A)_n$.

- 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 6.4 *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 6.5 *If a protocol is a t-resilient PP protocol then it is a t-resilient SPP protocol.*