7 Distributed Cryptography

7.1 Motivation

Distributed cryptography spreads the operation of a cryptosystem among a group of *parties* (*servers*) in a fault-tolerant way [Des94]. We consider the threshold failure model with n parties, of which up to t are faulty; such distributed cryptosystems are called *threshold cryptosystems*.

Distributed cryptosystems are based on *secret sharing* and are typically known only for public-key cryptosystems because of their "nice" algebraic properties. Here we consider a *public-key cryptosystem* and a *digital signature scheme*.

7.2 Secret Sharing

Secret sharing forms the basis of threshold cryptography. In a $(t+1)$-out-of-n *secret sharing scheme*, a secret s, element of a finite field $\mathbb{F}_q$, is shared among n parties such that the cooperation of at least $t+1$ parties is needed to recover s. Any group of t or fewer parties should not get any information about s.

Algorithm 1. To *share* $s \in \mathbb{F}_q$, a *dealer* $P_d \notin \{P_1, \ldots, P_n\}$ chooses uniformly at random a polynomial $f(X) \in \mathbb{F}_q[X]$ of degree t subject to $f(0) = s$, generates *shares* $s_i = f(i)$, and sends s_i to P_i for $i = 1, \ldots, n$. To reconstruct s among a group of $t+1$ parties with indices $\mathcal{S}$, every party reveals its share and they publicly recover the secret by computing

$$s = f(0) = \sum_{i \in \mathcal{S}} \lambda_{0,i}^{\mathcal{S}} \, s_i,$$

where

$$\lambda_{0,i}^{\mathcal{S}} = \prod_{j \in \mathcal{S}, j \neq i} \frac{j}{j-i}$$

are the (easy-to-compute) Lagrange coefficients. The scheme has perfect security, i.e., the shares held by every group of t or fewer parties are statistically independent of s (as in a one-time pad).

Verifiable Secret Sharing. If the dealer P_d may also be faulty (i.e., malicious and actively deviating from the protocol), we need a *verifiable secret sharing (VSS)*, a fault-tolerant protocol to ensure that P_d distributes "consistent" shares in the sense that (1) if some party terminates the sharing successfully, then every other correct party eventually also terminates successfully, and (2) every group of parties qualified to recover the secret will recover the same value. VSS is an important building block for secure multi-party computation [Fel87, Ped92].

Distributed Key Generation. There are also *distributed key-generation protocols (DKG)* for generating a public key and a sharing of the corresponding secret key. They ensure that the corrupted parties learn no information about the secret key. Such protocols exist and have been implemented for the common public-key types, those based on discrete logarithms and on RSA. Usually these protocols work only in synchronous networks and tolerate a passive adversary. Under weaker assumptions (asynchrony and active adversary), they are less practical.

7.3 Threshold ElGamal Encryption

Discrete Logarithms. Let $G = < g >$ be a group of prime order q, such that g is a generator of G. The *discrete logarithm problem (DLP)* means, for a random $y \in G$, to compute $x \in \mathbb{Z}_q$ such that $y = g^x$. The *Diffie-Hellman problem (DHP)* is to compute $g^{x_1 x_2}$ from two random values $y_1 = g^{x_1}$ and $y_2 = g^{x_2}$.

It is conjectured that there exist groups in which solving the DLP and DHP is *hard*, for example, the multiplicative subgroup $G \subset \mathbb{Z}_p^*$ of order q, for some prime $p = mq + 1$ (recall that q is prime). For example, $|p| = 2048$ and $|q| = 256$ for 2048-bit discrete-logarithm-based cryptosystems, which is considered secure today. Using the language of complexity theory, to say that a problem is *hard* means that any *efficient* algorithm solves it only with *negligible* probability. (Formally, this is defined using complexity-theoretic notions [Gol04]: there is a *security parameter* k, an *efficient algorithm* is probabilistic and runs in time bounded by a fixed polynomial in k, and a *negligible function* is smaller than any polynomial fraction.)

Public-key Cryptosystems. A *public-key cryptosystem* is a triple $(\mathsf{K}, \mathsf{E}, \mathsf{D})$ of efficient algorithms. Algorithm K generates a pair of keys (pk, sk) and is probabilistic. The encryption algorithm E is probabilistic and the decryption algorithm D is (usually) deterministic; they have the property that for all (pk, sk) generated by K and for any plaintext message m, the probability that $\mathsf{D}(sk, \mathsf{E}(pk, m)) \neq m$ is negligible.

A public-key cryptosystem is *semantically secure* if no efficient adversary A can find two messages m_0 and m_1 such that A can distinguish their encryptions. More precisely, A runs in two stages and first outputs m_0 and m_1; then a random bit b is chosen and A is given $c = \mathsf{E}(pk, m_b)$; A can distinguish encryptions if it can guess b from c correctly with more than negligible probability. Semantic security provides security against a *passive* adversary, but not against an *active* one.

ElGamal Encryption. The *ElGamal* cryptosystem is based on the Diffie-Hellman problem: Key generation chooses a random secret key $x \in \mathbb{Z}_q$ and computes the public key as $y = g^x$. The encryption of $m \in \{0,1\}^k$ under public-key y is the tuple $(c_1, c_2) = (g^r, m \oplus H(y^r))$, computed using a randomly chosen $r \in \mathbb{Z}_q$ and a hash function $H : G \to \{0,1\}^k$. The decryption of a ciphertext (c_1, c_2) is $\hat{m} = H({c_1}^x) \oplus c_2$. One can easily verify that $\hat{m} = m$ because ${c_1}^x = g^{rx} = g^{xr} = y^r$, and therefore, the argument to H is the same in encryption and decryption. The scheme is considered secure against passive adversaries. (For actually proving that breaking semantic security is as hard as solving the DHP, one has to use the random-oracle model.)

Threshold ElGamal Encryption. The following $(t+1)$-out-of-n threshold ElGamal cryptosystem tolerates the passive corruption of $t < n/2$ parties.

Let the secret key x be *shared* among $P_1, \ldots, P_n$ using a polynomial f of degree t over $\mathbb{Z}_q$ such that P_i holds a share $x_i = f(i)$. The global public key $y = g^x$ is known to all parties, and encryption proceeds as in standard ElGamal above. For decryption, a client sends a decryption request containing c_1, c_2 to all parties. Upon receiving a decryption request, party P_i computes a *decryption share* $d_i = {c_1}^{x_i}$ and sends it to the client. Upon receiving decryption shares from a set of $t + 1$ parties with indices $\mathcal{S}$, the client computes the message as

$$m \;=\; H\Big(\prod_{i \in \mathcal{S}} {d_i}^{\lambda^{\mathcal{S}}_{0,i}}\Big) \oplus c_2.$$

This works because

$$\prod_{i \in \mathcal{S}} {d_i}^{\lambda^{\mathcal{S}}_{0,i}} \;=\; \prod_{i \in \mathcal{S}} {c_1}^{x_i \lambda^{\mathcal{S}}_{0,i}} \;=\; {c_1}^{\sum_{i \in \mathcal{S}} x_i \lambda^{\mathcal{S}}_{0,i}} \;=\; {c_1}^{x}$$

from the properties of Algorithm 1. Note that the decryption operation only requires the cooperation of $t + 1 \leq n - t$ parties.

This is an example of a *non-interactive* threshold cryptosystem, as no interaction among the parties is needed. It can also be made robust, i.e., secure against an active adversary [SG02]. Such threshold cryptosystems can easily be integrated in asynchronous distributed systems; but many threshold cryptosystems are only known under the stronger assumption of *synchronous* networks with broadcast.

7.4 Threshold RSA Signatures

Threshold versions of the RSA cryptosystem and the RSA signature scheme are more difficult to obtain than for discrete-logarithm-based schemes. The reason is that the order of the group, from which the secret exponents are drawn, must not be revealed.

Digital Signature Schemes. A digital signature scheme is a triple $(\mathsf{K}, \mathsf{S}, \mathsf{V})$ of efficient algorithms. The *key generation* algorithm K outputs a public key/private key pair (pk, sk). The signing algorithm S takes as input the private key and a message m, and produces a signature σ. The verification algorithm V takes the public key, a message m, and a putative signature σ, and outputs a bit that indicates whether it accepts or rejects the signature. The signature is *valid* for the message when V accepts. All signatures produced by the signing algorithm must be valid.

A digital signature scheme is secure against *existential forgery* if no efficient adversary A can output any message together with a valid signature that was not produced by the legitimate signer. More formally, A is given *pk* and is allowed to request signatures on a sequence of messages of its choice, where any message may depend on previously obtained signatures. If A can output a message whose signature it never requested, then the adversary has successfully *forged* a signature. A signature scheme is *secure* if any efficient A can forge a signature only with negligible probability.

RSA Signatures. Let $N = pq$ be the product of two primes of approximately equal length. For example, $|p| = |q| \approx 1024$ in the case of RSA with 2048 bits, which is considered secure today. The group $\mathbb{Z}_N^*$ has order $\varphi(N) = (p-1)(q-1)$; it is believed that the only way to compute $\varphi(N)$ requires knowledge of the prime factorization of N. RSA also uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$.

Algorithm K chooses two random primes p and q and a (potentially fixed) prime e. Then it computes $N = pq$ and $d \equiv e^{-1} \mod \varphi(N)$, and outputs $sk = d$ and $pk = (N, e)$.

To sign a message m, algorithm S computes $\sigma = H(m)^d$ in $\mathbb{Z}_N^*$, i.e., modulo N. The verification algorithm tests if a signature σ is valid for a message m by checking whether $\sigma^e \overset{?}{=} H(m)$ in $\mathbb{Z}_N^*$.

Threshold RSA Signatures. Given the number-theoretic structure of RSA, one cannot perform interpolation "in the exponent" as in the discrete-log setting because the order of the group, $\varphi(N)$, must remain secret.

A simple n-out-of-n threshold signature scheme can be obtained nevertheless, by using *additive sharing* of the private key over the *integers*. It provides security against a passive adversary. The dealer chooses random values $d_i \in \mathbb{Z}$ for $i = 1, \ldots, n$ such that $d \equiv \sum_{i=1}^n d_i \mod \varphi(N)$. In order not to reveal information about d or $\varphi(N)$, the d_i are chosen with bit length significantly larger than d, e.g., $|d_i| \approx |d| + 160$. This method hides d statistically.

To set up the scheme, the dealer generates an RSA key pair and shares d among $P_1, \ldots, P_n$ over the integers, such that P_i receives d_i.

To sign a message m, a client sends the request to all parties; a party P_i computes a *signature share* $\sigma_i = H(m)^{d_i}$ and returns σ_i to the client. From n received signature shares, the client computes the signature $\sigma = \prod_{i=1}^n \sigma_i$ in $\mathbb{Z}_N$. Note that

$$\sigma = \prod_{i=1}^{n} \sigma_i = \prod_{i=1}^{n} H(m)^{d_i} = H(m)^{\sum_{i=1}^n d_i} = H(m)^d$$

because $d \equiv \sum_{i=1}^n d_i \mod \varphi(N)$. Verification is the same as with ordinary RSA signatures.

The drawback of this scheme is that the cooperation of *all* n parties is required for signing because *additive* sharing is used. Nevertheless, it is also possible to use a polynomial sharing and to obtain a truly fault-tolerant RSA-based threshold signature scheme. Shoup's scheme [Sho00, GHKR08], for example, is robust, i.e., secure against an active adversary, and is also non-interactive, which makes it suitable for use in asynchronous distributed systems.

7.5 A Distributed Pseudo-Random Function

A *pseudo-random function (PRF)* $F_x : \{0,1\}^* \to \{0,1\}^k$ is parameterized by a secret key x (called the *seed*) and maps an arbitrary-length input string to a fixed-length, k-bit output string that looks random to anyone who does not know the secret key. More formally, the PRF is secure if any efficient adversary who queries an oracle with distinct inputs cannot tell, with better than negligible probability, whether the oracle responds to the queries by evaluating the PRF on a random seed (known only to the oracle) or whether the oracle responds every time with a k-bit string freshly chosen at random with uniform distribution [Gol04].

In practice one often implements a PRF by a block cipher with a secret key; distributed implementations, however, are only known for functions based on public-key cryptosystems. Cachin et al. [CKS05] describe the following *threshold PRF*, which is suitable for integration in distributed protocols.

Algorithm 2 ([CKS05]). The scheme uses a group $G = < g >$, in which the DLP is hard; Let x be a randomly chosen *seed* and define a $F_x : \{0,1\}^* \to \{0,1\}^k$ as

$$F_x(v) = H'\big(H(v)^x\big),$$

where $H : \{0,1\}^* \to G$ and $H' : G \to \{0,1\}^k$ are two hash functions. The family $F = \{F_x\}$ is a pseudorandom function, assuming the hardness of the DLP (which can proven formally when H is modeled as a so-called random oracle).

A *threshold PRF* can be obtained analogously to threshold ElGamal encryption. Let a trusted dealer choose the seed x and share it among the parties with $(t+1)$-out-of-n polynomial secret sharing, such that party P_i holds share x_i. When it is time to compute $F_x(v)$, every correct party P_i computes a function share $d_i = \big(H(v)\big)^{x_i}$ and releases d_i according to the protocol. Any $t+1$ correctly computed function shares, from parties with indices in a set $\mathcal{S}$, yield the value of the PRF,

$$F_x(v) \;=\; H'\Big(\prod_{i \in \mathcal{S}} d_i^{\lambda^{\mathcal{S}}_{0,i}}\Big).$$

Writing $h = H(v)$, this is correct because (computed in G)

$$\prod_{i \in \mathcal{S}} d_i^{\lambda^{\mathcal{S}}_{0,i}} \;=\; \prod_{i \in \mathcal{S}} h^{x_i \lambda^{\mathcal{S}}_{0,i}} \;=\; h^{\sum_{i \in \mathcal{S}} x_i \lambda^{\mathcal{S}}_{0,i}} \;=\; h^x.$$

One can show that this does not leak information about x, under the assumption that the DLP is hard.

The threshold PRF can implement many instances of a *common coin* primitive, in order to output a sequence of shared coins *coin*.0, *coin*.1, ..., as needed by randomized (Byzantine) consensus protocols [CGR11, Sections 5.5 and 5.7]. Concretely, one sets the output length of the PRF to $k = 1$ and lets the input string v for the coin instance of round r be equal to $v = \mathit{coin}.r$, where the identifier of the instance is represented as a bit string. As usual, the identifier *coin* must be unique for all such protocol instances, and it must also be contained in every message and included in all signatures.

The threshold pseudorandom function is non-interactive. This means that no interaction among the parties is needed to compute the function value. To implement the *release* operation of the common coin instance *coin*.r, every party computes its function share (d_i) and sends it to all others; then every party collects $t+1$ such shares and combines them to the coin output value $b = F_x(\mathit{coin}.r)$.

The threshold PRF as described here tolerates only a passive adversary, but one can easily make it robust against an active adversary by adding zero-knowledge proofs for the correctness of the function shares generated by the parties [CKS05].

7.6 Proactive Security

7.6.1 Model

A *proactively secure cryptosystem* is a threshold cryptosystem using a group of n parties, where the shares of the parties are periodically refreshed [HJJ+97]. Recall that in an $(t+1)$-out-of-n-threshold (public-key) cryptosystem, every party holds a share of the private key, which is generated using secret sharing with a polynomial of degree t. In order to execute a cryptographic operation, at least $t+1$ parties must collaborate, and up to t parties may be faulty. Moreover, executing the operation does not leak any information about one party's share to another party or require the parties to pool their shares.

For high-value keys with a long lifetime, however, there is a risk that an attack spreads through the system and over time affects all parties, although not all of them simultaneously.

For instance, an adversary may slowly break into one party after another over time. Even when such break-ins can be detected, the exposure of a key share to the adversary cannot be undone.

Proactively secure systems perform system rejuvenation steps periodically, in anticipation of successful break-ins, and render leaked key shares harmless, in order to eliminate the long-time exposure problem. To implement this, proactive cryptosystems periodically refresh the shares held by the parties, such that a share exposed in a particular period is useless to an adversary in subsequent periods. The renewed shares still correspond to the same long-term private key, so that the long-term public key can remain unchanged.

In this section we assume for simplicity that the parties are *synchronized* and have access to a *common clock* and to a *synchronous broadcast channel* (in contrast to the rest of the course, which uses an asynchronous system). Furthermore, the parties are connected by secure channels, i.e., they can send private and authenticated point-to-point messages.

Time is divided into *periods*, determined by the common clock (for example, one day). Each time period consists of two phases: (1) a short *refresh phase*, during which the parties carry out the refresh protocol so that they hold fresh shares afterwards; and (2) a long *computation phase*, where the parties execute operations of the cryptosystem.

Infected parties should be rebooted and re-initialized by a trusted agent (e.g., from a read-only device) after a corruption has been discovered. We assume the adversary cannot impersonate a disinfected party, and may no longer send messages in its name or receive messages addressed to it. Proactive cryptosystems tolerate up to t *corrupted parties during every period*, but all parties may be corrupted over multiple periods. A corrupted party that has been disinfected in one period will be correct in the subsequent period, and a corrupted party that has not been disinfected remains corrupted also in the subsequent period (in particular, it may participate in the refresh protocol). In order not to leak secrets from past periods, it must be possible for a party to erase information permanently.

7.6.2 Proactive Refresh Tolerating Passive Attacks

The proactive resharing protocol below offers privacy but no robustness. This means that the corrupted parties follow the protocol, but they try to obtain more information than they are entitled to and leak this information to an adversary. The same security notion was already used for the distributed ElGamal cryptosystem in Section 7.3.

Suppose the n parties hold a polynomial sharing of the private key x for a discrete logarithm-based cryptosystem with public key $y = g^x$.

Algorithm 3 (Proactive refresh [HJKY95]). At the begin of the refresh phase, every party P_i holds a share $x_i = f(i) = \sum_{k=0}^{t} f_k i^k$ from the previous period. The refresh protocol consists of two steps:

1. Every party P_i chooses uniformly at random a polynomial $a^{(i)}(X) \in \mathbb{F}_q[X]$ of degree t subject to $a^{(i)}(0) = 0$. It generates shares $r_{ij} = a^{(i)}(j)$ for $j = 1, \ldots, n$, and sends r_{ij} to P_j as a private point-to-point message. (Observe that P_i essentially acts as the dealer to share the value 0 using polynomial secret sharing.)

2. After receiving n shares r_{ji}, for $j = 1, \ldots, n$, party P_i computes its new share in $\mathbb{Z}_q$ as
$$x'_i = x_i + \sum_{j=1}^{n} r_{ji}.$$

Then it *erases* all variables except x'_i.

At the end of the refresh phase, P_i uses x'_i as its fresh share for the computation phase of the period.

Theorem 4. *Provided $n > 2t$, Algorithm 3 ensures that:*

1. *When the input shares $x_1, \dots, x_n$ are a $(t+1)$-out-of-n sharing of x, then the output shares $x'_1, \dots, x'_n$ are also a $(t+1)$-out-of-n sharing of x.*

2. *An adversary that observes the secrets of at most t parties in every period learns no information about the private key x.*

Proof (sketch). To show the first condition (correctness), observe that every party P_i basically shares the value 0 as the dealer in a polynomial secret sharing scheme using $a^{(i)}(X)$. Given the sharing polynomial $f(X)$ of the previous period, the addition of all shares produces a new sharing polynomial

$$f'(X) = f(X) + \sum_{j=1}^{n} a^{(j)}(X).$$

And since $a^{(j)}(0) = 0$ for all $j = 1, \dots, n$, it holds $f'(0) = f(0)$.

In other words, suppose that there is a group $\mathcal{S}$ of $t+1$ parties that could recover the private key from the previous sharing as $x = f(0) = \sum_{i \in \mathcal{S}} \lambda^{\mathcal{S}}_{0,i} x_i$, with Lagrange coefficients $\lambda^{\mathcal{S}}_{0,i}$ for $i \in \mathcal{S}$. Then the recover operation from the new shares gives

$$\begin{aligned}\sum_{i \in \mathcal{S}} \lambda^{\mathcal{S}}_{0,i} x'_i &= \sum_{i \in \mathcal{S}} \lambda^{\mathcal{S}}_{0,i} \Big(x_i + \sum_{j=1}^{n} r_{ji} \Big) \\ &= \sum_{i \in \mathcal{S}} \lambda^{\mathcal{S}}_{0,i} x_i + \sum_{i \in \mathcal{S}} \lambda^{\mathcal{S}}_{0,i} \sum_{j=1}^{n} a^{(j)}(i) \;=\; x + \sum_{j=1}^{n} a^{(j)}(0) \;=\; x.\end{aligned}$$

To show the second condition (secrecy), suppose the adversary corrupts t_{prev} parties in the previous period *but not* in the current period (these parties have been disinfected), t_{both} parties in the previous period *and* in the current period (they remain corrupted during the refresh protocol), and t_{curr} parties *only* in the current period (they may be corrupted already during the refresh protocol). The assumption means that $t_{prev} + t_{both} \leq t$ and $t_{both} + t_{curr} \leq t$.

For every possible value of x, since the shares r_{ij} are sent privately and the adversary never learns more than t shares of any polynomial $a^{(i)}$ that is generated by a correct P_i, all information that it observes is consistent with x. Hence, it learns no information about x. □

7.6.3 More Robust Proactive Refresh

Algorithm 3 can be made *robust* so that it tolerates an active or Byzantine adversary. One problem with the above protocol is that a corrupted party in step 1 may send inconsistent share values r_{ij} or simply "share" a value $\neq 0$, so that the parties no longer hold a correct sharing of the private key. The extensions described by Herzberg et al. [HJKY95] and Gennaro et al. [GJKR07] use the mechanisms of VSS to prevent this attack.

Proactive cryptosystems for *asynchronous* networks also build on the principles presented here [CKLS02, ZSvR05].

References

[CGR11] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming (Second Edition)*, Springer, 2011.

[CKLS02] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, *Asynchronous verifiable secret sharing and proactive cryptosystems*, Proc. 9th ACM Conference on Computer and Communications Security (CCS), 2002, pp. 88–97.

[CKS05] C. Cachin, K. Kursawe, and V. Shoup, *Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography*, Journal of Cryptology **18** (2005), no. 3, 219–246.

[Des94] Y. Desmedt, *Threshold cryptography*, European Transactions on Telecommunications **5** (1994), no. 4, 449–457.

[Fel87] P. Feldman, *A practical scheme for non-interactive verifiable secret sharing*, Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS), 1987, pp. 427–437.

[GHKR08] R. Gennaro, S. Halevi, H. Krawczyk, and T. Rabin, *Threshold RSA for dynamic and ad-hoc groups*, Advances in Cryptology: Eurocrypt 2008 (N. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, pp. 88–107.

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, *Secure distributed key generation for discrete-log based cryptosystems*, Journal of Cryptology **20** (2007), 51–83.

[Gol04] O. Goldreich, *Foundations of cryptography*, vol. I & II, Cambridge University Press, 2001–2004.

[HJJ$^+$97] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, *Proactive public key and signature systems*, Proc. 4th ACM Conference on Computer and Communications Security (CCS), 1997.

[HJKY95] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, *Proactive secret sharing or how to cope with perpetual leakage*, Advances in Cryptology: CRYPTO '95 (D. Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer, 1995, pp. 339–352.

[Ped92] T. P. Pedersen, *Non-interactive and information-theoretic secure verifiable secret sharing*, Advances in Cryptology: CRYPTO '91 (J. Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1992, pp. 129–140.

[SG02] V. Shoup and R. Gennaro, *Securing threshold cryptosystems against chosen ciphertext attack*, Journal of Cryptology **15** (2002), no. 2, 75–96.

[Sho00] V. Shoup, *Practical threshold signatures*, Advances in Cryptology: EUROCRYPT 2000 (B. Preneel, ed.), Lecture Notes in Computer Science, vol. 1087, Springer, 2000, pp. 207–220.

[ZSvR05] L. Zhou, F. B. Schneider, and R. van Renesse, *APSS: Proactive secret sharing in asynchronous systems*, ACM Transactions on Information and System Security **8** (2005), no. 3, 259–286.