

5 Distributed Cryptography

5.1 Motivation

Distributed cryptography spreads the operation of a cryptosystem among a group of *servers* (or *parties*) in a fault-tolerant way [Des94]. We consider only the threshold failure model with n servers, 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*.

5.2 Adversarial Faults

Malicious activity requires a refined classification of faults.

A. random, non-adversarial faults

- 1) fail-stop [servers may crash, but crashes can be detected remotely; as with a perfect failure detector]
- 2) crash(-silent) [servers may crash, not detectable remotely]
- 3) crash-recovery [servers may crash and recover afterwards; using stable storage]
- 4) omission [servers may omit receiving, processing, or sending of messages]

B. adversarial and malicious faults

- 1) passive (or honest-but-curious or semi-honest model) [no deviation from specification, adversary with read-only privileges]
- 2) passive with crashes [a crash is the only deviation from specification]
- 3) active (or fully Byzantine) [arbitrary behavior]

This classification is intended for asynchronous distributed systems, but most concepts apply also in other contexts. So far, we have mostly used **A2** crash failures; in this section, we address **B1** and **B2**, with a passive adversary. Note that this classification is not one-dimensional, although there are some strict inclusion relations, like **A1-A2-B2-B3** and **A2-A4-B3**.

Additional, orthogonal assumptions for **B** adversarial and malicious faults:

a. setup

- i) trusted dealer [for generation and initial distribution of keys]

ii) active (or fully Byzantine) [arbitrary behavior]

b. adversary power

i) computationally bounded (running time bounded by a polynomial)

ii) unbounded

With adversarial faults, non-faulty servers are typically called *honest* and faulty servers are typically called *corrupted*, to emphasize that a malicious adversary may be coordinating the faulty parties.

5.3 Secret Sharing

Secret sharing forms the basis of threshold cryptography. A secret is *shared* among n parties such that the cooperation of at least $t + 1$ parties is needed to recover s .

Algorithm 5.1. To share $s \in \mathbb{F}_q$, a dealer $P_d \notin \{P_1, \dots, P_n\}$ 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 . To recover s among a group of $t + 1$ servers with indices \mathcal{S} , every server reveals its share and they publicly recover the secret

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

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 servers are statistically independent of s (as in a one-time pad).

Verifiable Secret Sharing. If the dealer P_d may also be faulty (malicious), we need a *verifiable secret sharing (VSS)*, a fault-tolerant protocol to ensure that P_d distributes “consistent” shares, which define a unique secret. VSS is an important building block for secure multi-party computation.

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 must 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, discrete logarithm and RSA. Usually they assume synchronous networks and passive adversaries. With weaker assumptions (active adversary), they are less practical, however.

5.4 Threshold ElGamal Encryption

Discrete Logarithms. Let $G = \langle g \rangle$ 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 random $y_1 = g^{x_1}$ 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 $p = mq + 1$, where $|p| = 1024$ and $|q| = 160$ (recall that q is prime). 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 a probabilistic that 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 (K, E, D) of efficient algorithms. Algorithm K generates a pair of keys (pk, sk) . The encryption and decryption algorithms, E and D , have the property that for all (pk, sk) generated by K and for any plaintext message m , the probability that $D(sk, E(pk, m)) \neq m$ 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 it can distinguish encryptions of them. 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 $c = E(pk, m_b)$; 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 $H : G \rightarrow \{0, 1\}^k$. 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 widely considered to be 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 threshold ElGamal cryptosystem tolerates the passive corruption of $t < n/2$ parties.

Let the secret key x be *shared* among P_1, \dots, 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 public key $y = g^x$ is global and known to all parties (and clients), and encryption is as in ElGamal above. For decryption, a client sends a decryption request containing c_1, c_2 to all servers. Upon receiving a decryption request, server 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 $t + 1$ servers with indices \mathcal{S} , the client computes the message as

$$m = H\left(\prod_{i \in \mathcal{S}} d_i^{\lambda_{0,i}^S}\right) \oplus c_2.$$

This works because

$$\prod_{i \in \mathcal{S}} d_i^{\lambda_{0,i}^S} = \prod_{i \in \mathcal{S}} c_i^{x_i \lambda_{0,i}^S} = c_i^{\sum_{i \in \mathcal{S}} x_i \lambda_{0,i}^S} = c_i^x$$

from the properties of Algorithm 5.1. Note that the decryption operation only requires the cooperation of $n - t$ servers.

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.

5.5 Threshold RSA Signatures

Threshold versions of the RSA cryptosystem and 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 $(\mathbf{K}, \mathbf{S}, \mathbf{V})$ of efficient algorithms. The *key generation* algorithm \mathbf{K} outputs a public key/private key pair (pk, sk) . The signing algorithm \mathbf{S} takes as input the private key and a message m , and produces a signature σ . The verification algorithm \mathbf{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. A signature is considered *valid* when \mathbf{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. 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, e.g., $|p| = |q| \approx 1024$. The group \mathbb{Z}_N^* has order $\varphi(N) = (p - 1)(q - 1)$, which can only be computed if the prime factorization of N is known. The specification of the scheme includes a hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_N$.

Algorithm \mathbf{K} chooses two random primes p and q and a (potentially fixed) prime e . Then it computes $N = pq$ and $d \equiv e^{-1} \pmod{\varphi(N)}$, 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 $\sigma^e \stackrel{?}{=} m$ in \mathbb{Z}_N .

Threshold RSA Signatures. In the RSA setting, one cannot perform interpolation “in the exponent” as in the discrete-log setting because the order of the group, $\varphi(N)$, must be secret.

A simple 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 $d_i \in \mathbb{Z}$ such that $d \equiv \sum_{i=1}^n d_i \pmod{\varphi(N)}$. 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, \dots, P_n over the integers, such that P_i received d_i .

To sign a message m , a client sends the request to all servers; a server 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 $d \equiv \sum_{i=1}^n d_i \pmod{\varphi(N)}$. Verification is the same as above.

The drawback of this scheme is that the cooperation of *all* n servers 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 [Sho00]. Shoup’s scheme, in addition, is robust, i.e., secure against an active adversary, and is also non-interactive, which makes it suitable for use in asynchronous distributed systems.

References

- [Des94] Y. Desmedt, *Threshold cryptography*, European Transactions on Telecommunications **5** (1994), no. 4, 449–457.
- [Gol04] O. Goldreich, *Foundations of cryptography*, vol. I & II, Cambridge University Press, 2001–2004.
- [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.