Notes on Cryptography and Computer Security

Mahdi Zamani
Visa Research, Palo Alto, CA
mzamani@visa.com

1 Introduction

In provable security, we prove that no adversary exists that can break the security of a scheme. To do this, we first need to define *notions of security* that clearly define adversary's goals and how much power and information it is given. We also need to choose our *proof models*, which clearly define the set of assumptions that help us proof the security of our algorithms. One also needs to define a *communication model*, which describes how the communication channels used by participants in a protocol behave. In this section, we describe various such models that are used frequently in cryptography and distributed algorithms communities.

1.1 Notions of Security

In terms of its computational power, an adversary can be categorized as one of the following types:

- **Computationally-bounded.** An adversary who is capable of only polynomial-time computation. A model with this type of adversary provides computational security.
- **Computationally-unbounded.** An adversary who has *unknown* (not necessarily infinite) amounts of resources. Thus, any algorithm in this model cannot assume any bound on the adversary's computational powers and it must maintain its security with *any* amount of computational power the adversary may have.

In terms of corruption abilities, an adversary can be categorized as one of the following types:

- **Fail-stop (*a.k.a.,* non-adversarial fault).** The adversary can only cause the parties to crash randomly.
- **Passive (*a.k.a.,* semi-honest or honest-but-curious).** The adversary can read the internal state of the corrupted players and their communication, trying to obtain some information he is not entitled to. This is called *eavesdropping*. Passive model is the weakest adversarial model.
- **Active (*a.k.a.,* malicious or Byzantine).** The adversary can additionally make the corrupted parties deviate from the protocol specification in order to falsify the outcome of the protocol. Such This is the strongest adversarial model.
- **Semi-malicious.** The adversary follows the protocol execution (similar to a semi-honest adversary), but can choose its random coins (and inputs) in any arbitrary manner.

- **Covert.** The adversary can make the corrupted parties deviate arbitrarily from the protocol specification, but only if this deviation cannot be *detected* [AL10]. Covert model is similar to the malicious model except in that it assumes the adversary and all its corrupted parties will be detected once at least one corrupted party is detected so the adversary is limited to only *covert* (hidden) attacks. One disadvantage of this model is that it cannot distinguish between fail-stop faults and non-participation attacks (if possible).
- **Rushing.** The adversary may delay sending the messages of the corrupted parties in any given round until *after* the honest parties send their messages in that round. As a result, the messages of the corrupted parties in every round may depend on the messages of the honest parties in that round. The standard definition of secure computation assumes a rushing adversary.

In terms of when the set of corrupted parties are chosen, an adversary can be categorized as one of the following types:

- **Static.** The adversary is restricted to choose its set of dishonest parties at the start of the protocol and cannot change this set later on.
- **Adaptive.** The adversary can choose its set of dishonest parties at any moment during the execution of the protocol.

With respect to the adversary's computational power, the security provided by a secure algorithm can be categorized as one of the following:

- **Information-theoretic security (perfect security).** An algorithm is perfectly-secure if given a ciphertext, every message in the message space is exactly as likely to be the plaintext, i.e., the plaintext is independent of the ciphertext. This implies that even a computationally-unbounded adversary cannot learn anything about the plaintext. Such an algorithm is not vulnerable to future developments of quantum computing. A symmetric encryption scheme is said to be perfectly-secure if for any key k, plaintexts P_1 and P_2, and ciphertext C,

  $$\Pr\Big[\mathrm{Enc}_k(P_1) = C\Big] = \Pr\Big[\mathrm{Enc}_k(P_2) = C\Big].$$

  This is sometimes called *unconditional security*, because the secure algorithm makes no condition (assumption) about the resources (computing power, memory, etc) available to the adversary so he can at best make a random guess to break the security of the algorithm. It is impossible for a perfectly-secure encryption scheme to have key sizes smaller than message sizes.
- **Statistical Security.** A slightly weaker notion of perfect security, where the adversary is given a small *advantage* (usually denoted by ϵ) in breaking the security of the algorithm than a purely random guess. Informally speaking, an encryption scheme is *ϵ-statistically secure* if the probability that the adversary guesses the correct message from a ciphertext is at most $1/2 + \epsilon$, for some small positive ϵ. In other words, for any key k, every pair of plaintexts P_1 and P_2, ciphertext C, and sample space Ω, the distributions $\mathrm{Enc}_k(P_1)$ and $\mathrm{Enc}_k(P_2)$ are within at most *ϵ-statistical distance* of each other, i.e.,

  $$\max_{\Omega}\Big|\Pr\Big[\mathrm{Enc}_k(P_1) = C\Big] - \Pr\Big[\mathrm{Enc}_k(P_2) = C\Big]\Big| \leq \epsilon.$$
- **Computational (cryptographic) security.** The information-theoretic notion of security is sometimes too strong (and hence inefficient) to be useful in practice. Computational security asks only that no polynomial-time adversary can tell which of the messages that could potentially be plaintexts corresponding to a ciphertext is more likely than the other to be the actual plaintext. In other words, the key space is big enough to make brute-force attacks impossible for such an adversary.

1.2 Security Proof Models

A security proof model describes the required assumptions for the proof of security of a secure algorithm to hold. The following are the most useful security proof models:

- **Strongest security model.** An algorithm is *secure in the strongest model* if it does not make any assumption for its proof of security. In other words, it proves that given whatever information and computational power an adversary desires, it is incapable of breaking the scheme even in the slightest way.
- **Standard model (*a.k.a.,* plain model).** An algorithm is *secure in the standard model* if the only assumption it makes for its proof of security is the computational power of the adversary (called the standard assumption).
- **Random oracle model.** A random oracle is an abstract black-box that generates truly random numbers. An algorithm is *secure in the random oracle model* if it assumes the existence of a random oracle for its proof of security.
- **Common reference string (CRS) model.** An algorithm is secure in the CRS model if it assumes that all parties have access to a common random string taken from a predetermined distribution.
- **Public key infrastructure (PKI) model.** An algorithm is secure in the PKI model if it assumes identities of all parties can be reliably verified using techniques of public-key cryptography (PKC). In other words, an algorithm secure in the PKI model assumes that the channels are already authenticated via PKC.
- **Circular security model.** Circular security describes the way that an encryption scheme behaves when it is asked to encrypt its own secret key. In theory, that behavior can be pretty unpredictable as such an encryption scheme might become completely insecure when it is asked to encrypt its own secret key (a great post by Matthew Green [Gre12] on circular security). It is not known whether circular security is possible under general assumptions.

1.3 Communication Models

A *secure channel* is probably the most basic element of cryptography. It is a communication stream between two parties (traditionally called *Alice* and *Bob*) with the following orthogonal requirements:

- **Authentication.** Ensures senders and receivers are who they claim to be.
- **Confidentiality.** Ensures that data is read only by authorized users.
- **Data integrity.** Ensures that data is not changed from source to destination.

With respect to how messages are delivered by the channel, the following communication models are often considered:

- **Fully-synchronous model.** This model assumes that all messages sent over the communication channels are delivered to their recipients *at the same time* and the messages cannot be delayed at all. This assumption is often considered to be unrealistic in practice since it assumes the existence of a global synchronization clock which is extremely hard (practically impossible) to implement in a distributed system.

- **Partially-synchronous model.** Also called *semi-synchronous* or *weakly-synchronous*, this model assumes that the messages sent are received by their recipients within some fixed time bound. In other words, while the messages may be delayed arbitrarily, they are guaranteed to be delivered within the time bound. In the Internet, messages are delivered in a partially synchronous manner. In cryptography, this is often modeled using a *rushing adversary*, who can wait to receive the honest parties' messages sent in a round and then decide what to send in the *same* round.

- **Asynchronous model.** Messages sent by parties may be arbitrarily delayed and no bound is assumed on the amount of time that it takes for the messages to be delivered. However, to show that the protocol terminates after some finite amount of time, all messages are often assumed to be delivered *eventually* after some bounded but unknown amount of time. Some asynchronous protocols make the extra assumption of some *synchronization point* in the course of the protocol run.

2 Groups and Fields

Most cryptographic algorithms are designed based on various properties of groups. In this section, we briefly review basics of the group theory.

2.1 Groups

A group is a set, G, with an operation that combines any two of its elements to form a third element also in the set. The set of integers together with addition is a group denoted by $\mathbb{Z}^+$ or simply $\mathbb{Z}$. Such a group is called an *additive group*. If instead of addition multiplication is considered, we get a *multiplicative group*. All elements of a multiplicative group must be invertible so zero is not a member of such a group. The multiplicative group of integers is denoted by $\mathbb{Z}^* = \{1, 2, 3...\}$.

A *finite group*, G_N, is a group that has a finite number of elements N, which is called the *order* of the group. The additive group of integers modulo n is $\mathbb{Z}_n = \{0, 1, 2, ..., n-1\}$ and the multiplicative group of integers modulo n is $\mathbb{Z}_n^* = \{x \in \mathbb{Z}_n : \gcd(x, n) = 1\}$, which is the set of all invertible elements of $\mathbb{Z}_n$. For example, $\mathbb{Z}_{12}^* = \{1, 5, 7, 11\}$. Obviously, if p is a prime then $\mathbb{Z}_p^* = \{1, 2, ..., p-1\}$.

Definition. The *modular multiplicative inverse* of an integer a modulo m is an integer denoted by a^{-1} such that $aa^{-1} \bmod m = 1$. This inverse exists if and only if a and m are *coprime*, i.e., $\gcd(a, m) = 1$.

A multiplicative group G is called *cyclic* if there exists an element $g \in G$ (called a *generator*) such that $G = \{g^n \mid n \text{ is an integer}\}$. The cyclic group $G = \{g^0, g^1, g^2, g^3\}$, where $g^4 = g^0$ is essentially the same as the additive group $\mathbb{Z}_4 = \{0, 1, 2, 3\}$. As an example, we say *3 is a generator modulo 7* because $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\}$, which is $\mathbb{Z}_7^*$. Note that not every element of $\mathbb{Z}_7$ is a generator. For example, 2 is not a generator modulo 7 because $\{2^0, 2^1, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\}$ meaning that it does not generate all elements of $\mathbb{Z}_7^*$.

Definition. Let p be a prime and g be a generator of the multiplicative group $\mathbb{Z}_p$. The *order* of g modulo p denoted by $\text{ord}_p(g)$ is the smallest integer q such that $g^q \bmod p = 1$. Based on a special case of Lagrange's theorem, $\text{ord}_p(g)$ always divides $p-1$.

2.2 Rings

A ring is a set R equipped with two binary operations $+$ and $\times$ called *addition* and *multiplication*, that map every pair of elements of R to a unique element of R. A ring with set R is sometimes denoted by $(R, +, \times)$. The most familiar example of a ring is the set of all integers $\mathbb{Z}$. Other examples are

- $2\mathbb{Z}$ that consists of all even integers.
- $\mathbb{Z}[x]$ that consists of all integer polynomials[1] in variable x.
- $\mathbb{Z}/4\mathbb{Z}$ or $\mathbb{Z}_4$ that consists of the numbers $0, 1, 2, 3$.

$\mathbb{Z}/4\mathbb{Z}$ is an example of a *quotient ring* (or *factoring ring*). Informally, if R_1 and R_2 are two rings, then R_1/R_2 is a quotient ring, which is a simplified version of R_1 constructed by ignoring (zeroing out) the elements of R_2. The quotient ring $\mathbb{Z}/n\mathbb{Z}$ for any integer n, is equivalent to the finite field $\mathbb{F}_n$. Modular arithmetic is the arithmetic in this quotient ring.

2.3 Finite Fields

A *field* is a commutative ring[2] that contains a multiplicative inverse for every nonzero element. A *finite field* (*Galois field*) is a field that contains a finite number of elements (called *order* of the field). The order of a finite field is always a prime p or a power of a prime p^n. The finite field with p elements is denoted by $GF(p)$ or $\mathbb{F}_p$, which is simply the ring of integers modulo p. Elements of $GF(p^n)$, where n is a positive integer may be represented as polynomials of degree less than n over $GF(p)$. Operations are then performed modulo R, which is an irreducible polynomial of degree n over $GF(p)$.

3 Hardness Assumptions

Mathematical problems are used to construct a cryptosystem. These problems are usually hard to solve unless one has extra information. Such information can act as a key for decrypting ciphertext generated based on the hard problem. Some examples of such hard problems are *integer factorization problem*, *discrete logarithm problem*, and *lattice problems*. Integer factorization is the decomposition of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer. Integer factorization and discrete logarithm problems can be easily solved using a large enough quantum computer. Lattice-based cryptography is a *post-quantum cryptography* method, which refers to cryptosystems that are not breakable even using a quantum computer.

One-Way Functions and Trapdoors. A *one-way function* is a function f, which is *easy* to compute on every input x but is *hard* to invert given any $f(x)$. A *trapdoor function* is a function that is easy to perform one way, but requires a secret key to perform the inverse calculation efficiently. All trapdoor functions are one-way functions, but not all one-way functions are trapdoors, because not all one-way functions require a special key to compute the inverse. For example, hash functions are one-way functions, but they are not trapdoor functions as they are not reversible at all. It is not known if perfect one-way functions exist because their existence implies that $P \neq NP$. Note that a one-way function must be hard to invert, even if the entire function definition is known and fixed. However, a *one-time pad function*, such as $f(x) = x + r$ on a finite field with r being a random element, is changed in every round. So, such a function is not a one-way function.

Discrete Logarithm Problem (DLP). Given g and g^x in a group G, computing x is computationally infeasible. Consider the equation

$$g^x \bmod N = h,$$

[1] i.e., polynomials with integer coefficients

[2] A ring in which the multiplication operation is commutative.

where g and h are elements of a finite cyclic group G. Given g, x, and N, it is easy to find h, but it is hard to find x from g, h, and N. Many algorithms are based on the difficulty of finding $\log_g h$. Diffie-Hellman key agreement protocol [DH76] (see section 5.1) is based on the hardness of discrete logarithm problem. Sometime only when $G = \mathbb{Z}_p^*$ for prime p, this problem is called DLP and if G is any cyclic group, then it is called the *generalized DLP* or *GDLP*.

Computational Diffie-Hellman (CDH) Problem. Given g, g^a, and g^b in a group G, computing g^{ab} is computationally infeasible. It is proven that CDH $\leq_p$ DLP [Ver13], i.e., solving CDH is at most as hard as solving DLP. In other words, any polynomial algorithm for solving DLP can also be used to solve CDH (polynomial reduction).

Decision Diffie-Hellman (DDH) Problem. Given h, g^a, and g^b in a group G, determining whether $h = g^{ab}$ is computationally infeasible. It is proven that DDH $\leq_p$ CDH [Ver13].

***t*-Strong Diffie-Hellman (*t*-SDH) Problem.** Given $g, g^a, \ldots, g^{a^t}$ in a group G, finding $c \in \mathbb{Z}_p$ and $g^{1/(a+c)}$ are computationally infeasible. It is proven that t-SDH $\leq_p$ DLP [Ver13].

Lattice Problems. Lattice is a set of points in n-dimensional space with a periodic structure. A lattice is an additive subgroup of R^n defined as a set of linearly-independent vectors such that their linear combination can generate all points in a given space. Three dimensional lattices occur naturally in crystals. *Worst-case lattice problems* means that breaking the cryptographic construction is provably at least as hard as solving several lattice problems in the worst case. Two famous lattice problems are the *Shortest Vector Problem (SVP)* and the *Closest Vector Problem (CVP)*. Algorithms based the hardness of these problems are called *post-quantum algorithms* as they cannot be broken in feasible time even by a quantum computer. There are non-lattice problems that are as hardness lattice problem. One example is the *Learning with Errors (LWE) problem*, which asks to recover a secret s given a sequence of "approximate" random linear equations on s. A special case of LWE, which results in practical key sizes and good hardness is called *ring-LWE*.

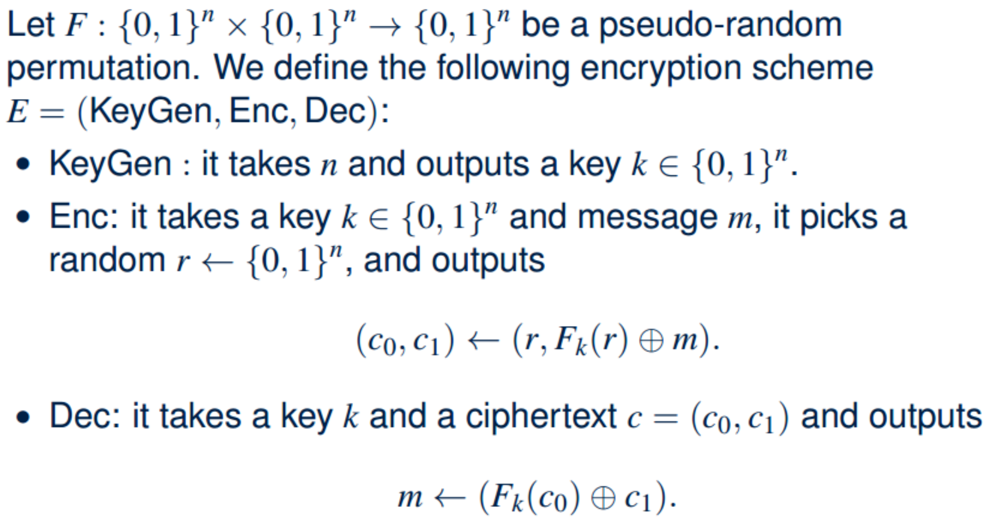
4 Symmetric Key Encryption

Secure Key Exchange. Alice and Bob agree on a *shared secret key* (i.e., both have the same key) that is later used to implement a secure channel. A *key* is a piece of information that allows only those that hold it to encode and decode a message. In most encryption systems, only secrecy of the keys provides security.

4.1 Pseudorandom Generation

Pseudo-Random Generator (PRG). An efficient, deterministic algorithm that expands a short, uniform seed into a longer, pseudo-random output. Do PRGs exist? Not if $NP = P$. An example of a non-crypto PRG is the linear congruential generator: $X_{n+1} = aX_n + b \mod m$, where a, b, m are constants and X_0 is the seed. Cryptographically-suitable PRGs continuously add entropy to their internal state. An example of entropy source is timing (hardware interrupts, e.g. keyboard, mouse, etc.).

Pseudo-Random Function (PRF). It is a generalization of the notion of PRGs, where a "random-looking" function is considered. A PRF is a *keyed function* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. Once we fix the secret key



Let $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a pseudo-random permutation. We define the following encryption scheme $E = (\text{KeyGen}, \text{Enc}, \text{Dec})$:

- KeyGen : it takes n and outputs a key $k \in \{0,1\}^n$.
- Enc: it takes a key $k \in \{0,1\}^n$ and message m, it picks a random $r \leftarrow \{0,1\}^n$, and outputs

$$(c_0, c_1) \leftarrow (r, F_k(r) \oplus m).$$

- Dec: it takes a key k and a ciphertext $c = (c_0, c_1)$ and outputs

$$m \leftarrow (F_k(c_0) \oplus c_1).$$

Figure 1: Encryption with PRP

k, we have $F_k : \{0,1\}^* \to \{0,1\}^*$. F is pseudo-random if the function F_k, for a uniform key k, is indistinguishable from a function chosen uniformly at random from the set of all functions with the same domain and range. The key is sometimes called the *seed.*

Pseudo-Random Permutation (PRP). A pseudo-random function $F : K \times X \to Y$ is an efficient pseudo-random permutation if the following hold:

1. F is one-to-one and $|X| = |Y|$;
2. F is deterministic and efficiently computable;
3. F^{-1} is efficiently computable.

In practice, we usually use $F : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. For example, in the AES encryption scheme $n = 128$ bits and $k = 128, 192$, or 256 bits. Encryption using PRP is shown in Figure 1.

4.2 Symmetric Cryptosystems

Stream Ciphers. A practical instantiation of a pseudo-random generator (PRG) typically with an encryption scheme that uses the pseudo-random bits generated by the PRG as the symmetric key to encrypt data. Every stream cipher has two main *deterministic* algorithms:

- $st_0 \leftarrow \text{Init}(s, V)$: takes a seed s and an optional initialization vector V and outputs an initial state st_0.
- $(y_i, st_i) \leftarrow \text{GetBits}(st_i)$: takes the i-th state information st_i and outputs a bit y and an updated state st_{i+1}.

Usually, a *keystream* is produced from the actual key. To encrypt a bit string m using a pseudo-random bit string k, we can compute $c = m \oplus k$, where $\oplus$ is the bitwise XOR operator. The ciphertext c can be decrypted using $m = c \oplus k$.

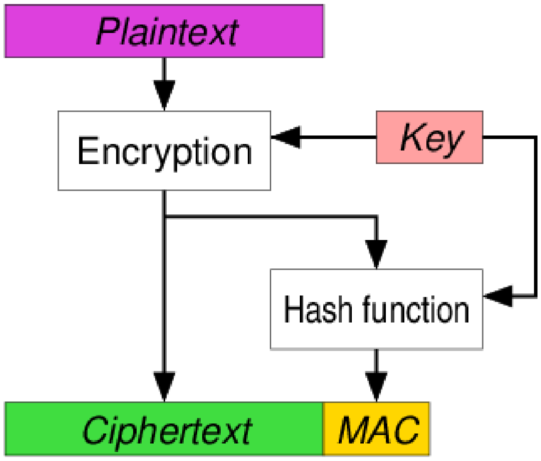


Figure 2: Authenticated encryption example

Authenticated Encryption (AE). Stream ciphers inherently provide no authentication and integrity-protection mechanism. This allows an adversary to conduct chosen-ciphertext and malicious bit-flipping attacks on the ciphertext. Therefore, a stream cipher is often used along message authentication codes (MACs) to protect against such attacks (see Figure 2). The result is often called an authenticated encryption system.

Block Ciphers. A block cipher is a deterministic algorithm for encrypting a block of data at once with the same key. AES is a symmetric block cipher.

Pohlig-Hellman Cryptosystem. Let p be a large prime such that $p - 1$ has a large prime factor. The Pohlig-Hellman cryptosystem is a symmetric cryptosystem, where the key is a pair $x, y \in \mathbb{Z}^*_{p-1}$ such that $xy \bmod (p-1) = 1$. A plaintext $m \in \mathbb{Z}_p$ is encrypted using $c \equiv m^x \bmod p$ and a ciphertext is decrypted by $m = c^y \bmod p$.

Deterministic Encryption. A cryptosystem which always generates the same ciphertext for a given plaintext and key. Examples are RSA and most block ciphers. Deterministic encryption is not semantically-secure as it can leak information about the plaintext to an eavesdropper if used multiple times; the adversary can simply recognize known ciphertexts and learn something every time that ciphertext is transmitted through statistical analysis or correlate ciphertexts with user actions.

Probabilistic Encryption. A cryptosystem that uses randomness in its encryption algorithm and thus produces different ciphertexts every time it is called (even with the same plaintext and key). To be semantically-secure, an encryption scheme has to be probabilistic. Examples of efficient probabilistic encryption algorithms are ElGamal and Paillier. A public key encryption scheme most likely needs to be probabilistic because the adversary can try encrypting each of his guesses under the recipient's public key, and compare each result to the target ciphertext.

4.3 Semantic Security

Semantic security (also called *computational indistinguishability*) guarantees that an adversary who sees the encryption of a message – drawn from a known or chosen message space – gains approximately the

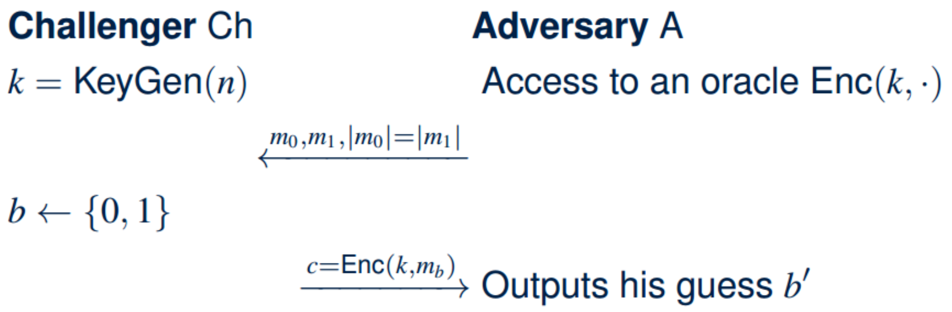

Challenger Ch
Adversary A
$k = \mathsf{KeyGen}(n)$
Access to an oracle $\mathsf{Enc}(k,\cdot)$
$m_0, m_1, |m_0| = |m_1|$
$b \leftarrow \{0,1\}$
$c = \mathsf{Enc}(k, m_b)$
Outputs his guess b'

Figure 3: CPA indistinguishability game

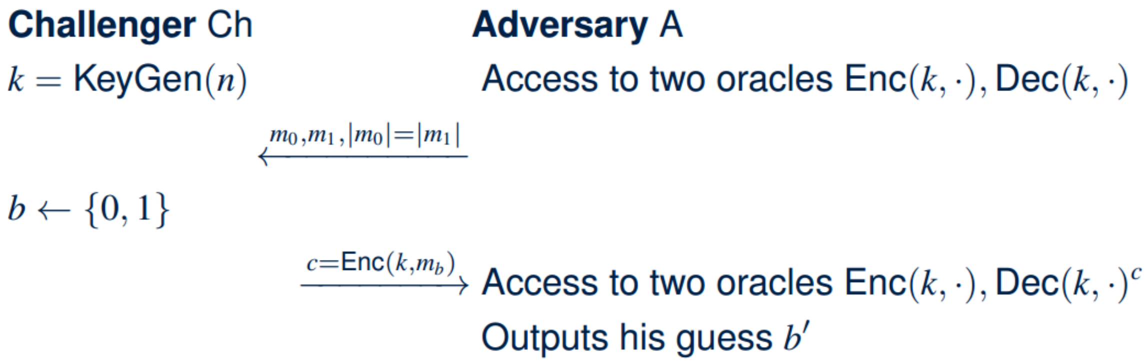

Challenger Ch
Adversary A
$k = \mathsf{KeyGen}(n)$
Access to two oracles $\mathsf{Enc}(k,\cdot), \mathsf{Dec}(k,\cdot)$
$m_0, m_1, |m_0| = |m_1|$
$b \leftarrow \{0,1\}$
$c = \mathsf{Enc}(k, m_b)$
Access to two oracles $\mathsf{Enc}(k,\cdot), \mathsf{Dec}(k,\cdot)^c$
Outputs his guess b'

Figure 4: CCA indistinguishability game

same amount of information as it would have obtained if it did not see the encryption in the first place.

Chosen-Plaintext Attack (CPA). An adversary enters one or more plaintexts (that he has chosen arbitrarily) into the system and obtains the resulting ciphertexts. From these pieces of information, he attempts to recover the secret key. In CPA, it is like the adversary has access to an oracle with the encryption function $\mathsf{Enc}(k, \cdot)$, where k is the secret key.

Chosen-Ciphertext Attack (CCA or CCA1). An adversary enters one or more ciphertexts (that he has chosen arbitrarily) into the system and obtains the resulting plaintexts. From these pieces of information, the adversary attempts to recover the secret key. In CCA, it is like the adversary has access to an oracle with the encryption function $\mathsf{Enc}(k, \cdot)$ and the decryption function $\mathsf{Dec}(k, \cdot)$, where k is the secret key.

Adaptive Chosen-Ciphertext Attack (CCA2). An adversary enters one or more ciphertexts (that he has chosen arbitrarily) into the system and obtains the resulting plaintexts. He then uses these pieces of information to choose subsequent ciphertexts that help him recover the secret key.

5 Public Key Cryptography

A public key encryption scheme acts like a *lock box*. Everyone is allowed to put something inside a box and lock it, but no one can open it except those who have the key. In public key encryption, a message is encrypted (*sealed*) with a *public key* so that it can only be decrypted using a *secret (private) key*. This is also known as *asymmetric cryptography* since the scheme requires two different keys: public key and private key.

If a message X is encrypted with a public key k, then anyone can verify a guess that $Y = X$ by checking whether $\mathrm{Enc}_k(Y) = \mathrm{Enc}_k(X)$. This threat is usually eliminated by attaching a large string of random bits to X before encrypting. Such a random value that is used only once is called a *nonce*.

5.1 Diffie-Hellman Key Exchange[3]

In 1976, Diffie and Hellman [DH76] proposed a method for two parties to securely agree on a public key. This scheme played an important role in the formation of public key cryptography in 1980s.

DH algorithm. Alice and Bob first agree on a large prime number p, and a generator (or base) g, where $0 < g < p$. Alice chooses a secret integer a (her private key) and calculates $g^a \bmod p$ (her public key). Bob chooses his private key b, and calculates his public key in the same way. Alice and Bob then send each other their public keys. Alice calculates $(g^b)^a \bmod p = g^{ab} \bmod p$ and Bob calculates $(g^a)^b \bmod p = g^{ab} \bmod p$. This means that they have agreed on a public key $g^{ab} \bmod p$ without learning each others private keys a and b unless they can solve the discrete logarithm problem, which is computationally infeasible.

Security of DH. The best known algorithm for solving the integer factorization problem (and also the discrete logarithm problem) is called *General Number Field Sieve (GNFS)*, which has been used to break ciphers with a 768-bit long prime p. In practice, p should have at least 1024 bits[4]. Also, $p - 1$ should have a large prime factor q, where $g^q \bmod p = 1$ (i.e., q be an order of g). This secures the algorithm against the Pohlig-Hellman attack. The Diffie-Hellman algorithm is vulnerable to Man-in-the-Middle attacks unless it is used along with a digital signatures scheme to authenticate sender of each message.

Elliptic Curve Diffie-Hellman (ECDH). Discrete logarithm problem (and thus DH algorithm) can be defined over other groups like elliptic curves, which result in much smaller key sizes with the same level of security. For example 3072-bit DH provides security comparable to ECDH-256.

5.2 RSA Cryptosystem

Encryption. Let p and q be two large primes (say each with 200 digits) and let e be an integer relatively prime to $(p - 1)(q - 1)$. Also, let M be the message to be encrypted and $n = pq$. The encryption of M, denoted by C, is calculated using

$$C = M^e \bmod n$$

Decryption. Let d be the decryption key, which is an inverse of e modulo $(p - 1)(q - 1)$. The ciphertext C can be decrypted using

$$C^d \equiv M \pmod{n}$$

Security of RSA. Given (n, e) as the public key, the adversary needs to factor $n = pq$ into p and q to compute $(p - 1)(q - 1)$, which can be used to compute the decryption key d from e. Hence, RSA is secure as long as $n = pq$ cannot be factored in a reasonable amount of time, which is currently the case for large

[3] Also called Diffie-Hellman key agreement.

[4] See RFC-3526 for a few parameters used in practice. See RFC-3766 for security strength comparison of DH, RSA, and DSA.

n's. This is called the *integer factorization problem.* RSA-768 (i.e., RSA with a 768-bit modulus denoted by n above) is the largest RSA that is broken so far. RSA-1024 is the most widely-used RSA.

Implementation notes. The large primes p and q can be found efficiently using a *primality test* algorithm. The encryption can be performed using a fast *modular exponentiation* algorithm.

5.3 ElGamal Cryptosystem

The Diffie-Hellman key exchange protocol [DH76] is can be used for generating shared secrets (usually used as crypto keys for encryption algorithms), but it cannot be used directly for encryption itself. The simplest way to use Diffie-Hellman as part of an encryption algorithm is to generate a shared one-time-pad that is XOR'd with the plaintext. The ElGamal encryption algorithm does basically this, the only difference is that it uses modular multiplication instead of XOR'ing to do the encryption once it has the shared one-time-pad. ElGamal cryptosystem is semantically-secure under CPA, but is not semantically-secure under CCA. For applications where CCA security is important one may use a CCA-secure cryptosystem such as Crame-Shoup [CS98].

Let p be a large prime such that $p - 1$ has a large prime factor q. Let $g \in \mathbb{Z}_p^*$ be an element of order q in $\mathbb{Z}_p^*$. In the ElGamal scheme, the private key is $x \in \{0, \dots, q-1\}$ and the public key is $g^x \bmod p$. A plaintext $m \in \mathbb{Z}_p$ is encrypted by the pair $(g^y \bmod p, m \cdot g^{xy} \bmod p)$, where $y \in \{0, \dots, q-1\}$ is a per-round (fresh) random value.

The ElGamal and the Pohlig-Hellman cryptosystems are *mutually commutative* meaning that a ciphertext encrypted under multiple ElGamal keys, multiple Pohlig-Hellman keys, or any mixture of the keys can be decrypted by the set of decryption keys in any order.

5.4 Distributed Key Generation

Distributed key generation (DKG) allows a set of n parties to jointly generate a pair of public and private keys in a such a way that the public key is output in the clear while the private key is shared by the parties via a threshold secret-sharing scheme such as that of [Sha79]. For discrete-log based threshold schemes, distributed key generation means a distributed generation (i.e., without a trusted dealer) of a Shamir secret sharing [Sha79] of a random value x as the private key, and $y = g^x$ as the public key.

Pedersen [Ped91a] proposed a simple and efficient DKG protocol that was later shown by Gennaro et al. [GJKR99] as failing to generate the private key (and the public key) with uniform distribution. They show that an adversary who compromises even two out of n parties can skew the distribution of the generated secret key. They also propose a new DKG protocol that fixes this problem by generating the private key with a uniform distribution. While Pedersen's protocol is non-interactive in the absence of faults, the protocol of Gennaro et al. requires two rounds of communication.

In a later work, Gennaro et al. [GJKR03] show that a certain type of discrete-log based threshold schemes can remain secure even if they use Pedersen's DKG protocol as a subprotocol. Namely, they show how to prove secure a threshold version of Schnorr's signature scheme [Sch90] instantiated with Pedersen's DKG protocol.

5.5 Verifiable Random Functions

First introduced by Micali, Rabin, and Vadhan in 1999 [**?**], a VRF is a pseudorandom function (PRF), where the party holding the secret key can produce a non-interactive proof that the function was evaluated correctly. The initial construction was inefficient. Dodis and Yampolskiy [?] propose an efficient VRF construction using bilinear maps.

5.6 Miscellaneous

Threshold Cryptography. In threshold cryptography, the message is encrypted using a public key and the corresponding secret key is shared among n parties. To decrypt the message, at least t parties are required to cooperate in the decryption algorithm. Such a scheme is called a (t, n)-threshold cryptosystem.

Anonymous Public-Key Encryption/Signatures. Encrypt/sign a message for reading/verifying by any member of an explicit anonymity set. For example, a group can use this to perform anonymous group communication, or a member of an organization's board of trustees might prove to be a member of the board without revealing which member he is (*anonymous authentication*). If the provided anonymity set contains only one public key, this reduces to conventional single-receiver public-key encryption/signatures.

6 Message Authentication and Integrity

Message authentication schemes guarantee that a message is sent by a valid sender. Message integrity schemes ensure that a message is protected from being manipulated or modified by an adversary. Confidentiality schemes (e.g. encryption and secret sharing) do not necessarily provide message integrity.

6.1 Message Authentication Codes

In order to verify integrity and authenticity of a message, an additional piece of information called MAC is generated using a MAC algorithm and sent along with the message to the receiver. The receiver then generates the MAC from the received message using the same MAC algorithm and verifies that both MACs match. This prevents the adversary from manipulating the messages.

The MAC algorithm requires a *MAC key* along with the message to generate the MAC. This method assumes the sender and receiver have the same MAC key as in the case of symmetric encryption. This is the difference between MACs and digital signatures as the latter is based on asymmetric encryption and provides *non-repudiation*, meaning that it proves that a message was signed by no one other than that holder of the secret key (i.e., its private key).

6.2 Digital Signatures

Digital signature (DS) algorithms are used to provide message authentication and integrity. A DS algorithm is usually based on public-key (asymmetric) cryptosystems (Section 5) and thus, usually consists of three algorithms

- $(pk, sk) = \mathsf{Keygen}()$
- $s = \mathsf{Sign}(m, sk)$
- $\mathsf{Verify}(m, s, pk)$,

where m is the message to be signed, s is the signature, and pk and sk are the public key and the secret key respectively generated by Keygen. If a message is digitally-signed via Sign, then Verify fails if and only if

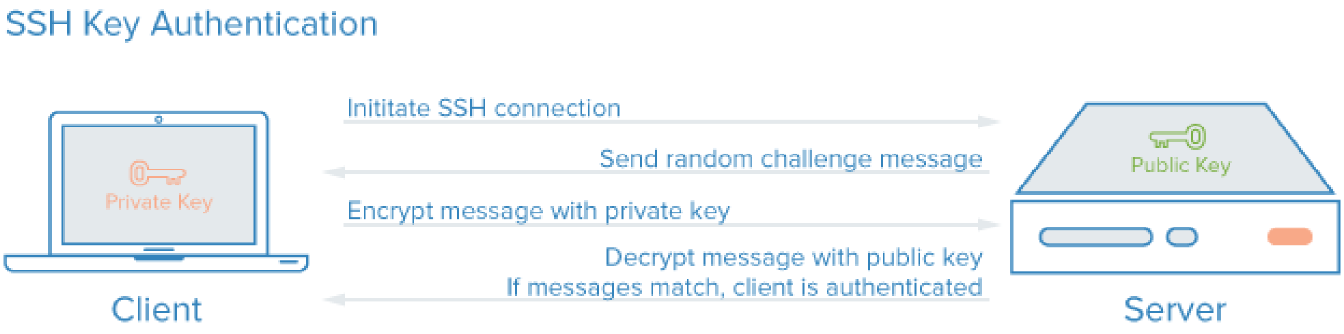

SSH Key Authentication
Private Key
Client
Inititate SSH connection
Send random challenge message
Encrypt message with private key
Decrypt message with public key
If messages match, client is authenticated
Public Key
Server

Figure 5: Public-key authentication in SSH.

- the message is changed after signing, and/or
- the message is signed with an invalid secret key (i.e., the secret key does not belong to the person the sender claims to be).

The idea behind DS schemes is to use a one-way function (Section 3): a message can be signed by computing the inverse of a one-way function using the secret key, and the signature can be verified by computing the forward direction. However, this type of DS is not semantically-secure because the adversary can use Verify to find a valid pair of a message and a signature.

Non-Repudiation. A *non-repudiable (non-transferable) signature* is a signature that is uniquely bound to the identity of the signer. Therefore, the signer cannot deny later that it signed the message. Non-repudiation can be achieved using public key cryptosystems. Singing a message using a symmetric encryption scheme such as a MAC does not provide non-repudiation because if Bob receives a message from Alice encrypted with a secret that is shared only between them, then Bob cannot prove to other parties that the message came from Alice. This is because there is no way to prove it was Alice and not Bob who signed the message with the shared secret. Such a signature is called a *transferable signature*.

6.2.1 Public-Key Authentication

A client proves its control of a private key associated with a public key by performing an operation which requires knowledge of the private key. This works by sending a signature created with the private key of the user. The signature is then verified using the public key of the user. Any public key algorithm may be offered for use in this type of authentication.

In SSH (see Figure 5), the user authentication protocol starts by the server sending a random challenge message M to the client. Then, the client encrypts M with its private key and sends it to the server. Finally, the server decrypts the message with the user's public key and if the result matches M, it sends a success message to the client. See [YL06] for more details about SSH public-key authentication.

6.3 Commitment Schemes

In a commitment scheme, a party commits to a secret value so that other parties ensure that he cannot change the value after he has committed to that value (called *consistency verification*). Every commitment scheme consists of two phases:

1. **Commit phase.** The sender sends a *commitment message* along with the secret to the receiver while ensuring,
    - The commitment message reveals nothing to the receiver (*hiding property*),
    - The commitment message is the only possible commitment value that the sender can compute from the message (*binding property*).
2. **Reveal phase.** The receiver wants to *open* (i.e., check) the commitment. The sender sends an *opening message* to the receiver, with which the receiver can perform a *check* that verifies consistency.

In other words, the commitment C to the value x is computed as $C = \mathsf{Commit}(x, open)$, where *open* is a random value that is also sent in the reveal phase as the opening message. The receiver then verifies the commitment using $\mathsf{CheckReveal}(C, x, open)$.

A commitment scheme is *perfectly-hiding* (*perfectly-binding*) if no unbounded adversary can break the hiding (binding) property described above. A commitment scheme is *computationally-hiding* (*computationally-binding*) if no polynomial-time adversary can break the hiding (binding) property. In general, a scheme cannot be both perfectly-hiding and perfectly-binding, because they are opposing principles.

A commitment scheme is *statistically-hiding* if the receiver can obtain some advantage from the commitment message but this advantage is provably exponentially small in the security parameter. Note that the receiver can still be unbounded in this type of commitment scheme.

Hiding property of commitment schemes is more crucial in some applications like verifiable secret sharing because it guarantees secrecy of the inputs. Binding property guarantees consistency, which is less critical than secrecy. For example in VSS, consistency violation at most results in incorrect sharing (or incorrect computation result in multiparty computation), which is more tolerable than revealing information about the secrets. Feldman [Fel87] introduces several commitment schemes all of which are computationally-binding and computationally-hiding. On the other hand, [Ped91b] gives a computationally-binding but perfectly-hiding commitment scheme, which can be used in unconditional models where the dealer is polynomial-time.

6.3.1 Pedersen Commitment Scheme

Pedersen [Ped91b] proposes a perfectly-hiding and computationally-binding commitment scheme[5], which is based on the difficulty of solving the discrete logarithm problem. Let p and q be primes such that q divides $p - 1$ and let $\mathbb{Z}_q$ be the additive group of integers modulo q and $\mathbb{Z}_p^*$ be the multiplicative group of integers modulo p. To commit to a value $s \in \mathbb{Z}_q$, the sender chooses $r \in \mathbb{Z}_q$ uniformly at random and computes the commitment

$$C(s, r) = g^s h^r,$$

where $g, h \in \mathbb{Z}_p^*$. The commitment can be opened (and then checked) once s and r are revealed. This scheme has the following homomorphic property

$$C(s_1, r_1) \cdot C(s_2, r_2) = C(s_1 + s_2, r_1 + r_2).$$

[5] Pedersen proposes his commitment scheme as a part of a verifiable secret sharing scheme (see section 13.1), which uses the commitment scheme to verify consistency of shares in Shamir's secret sharing scheme [Sha79]. Obviously, such a commitment scheme can be generally used in other applications like digital signatures and zero-knowledge proofs.

6.4 Anonymous Credentials

Anonymous credentials, first introduced by Chaum [Cha85], provide a mechanism for users of a system to anonymously prove assertions about themselves by issuing cryptographic credentials (or tokens) without disclosing any information but the fact that he possesses a valid credential. For example, an anonymous credential can be used to anonymously authenticate a user (i.e., prove his membership to an organization without revealing his identity) or to prove an assertion about the user in a privacy-preserving manner such as proving that the user is 18+ years old without revealing his exact date of birth.

It should be impossible (or practically hard) for an adversary to forge a credential for a user or link different credentials of the same user to each other when the user is assigned multiple credentials in different transactions. If the same credential can be used in multiple transactions, then it is called a *multiple-show credential*. Otherwise, it is called a *one-show credential*. In order to make multiple transactions unlinkable for a one-show credential, a set of credentials is issued and for each anonymous transaction, a new credential is used.

Anonymous credential systems are closely related to *blind signatures* [Cha83], zero-knowledge proofs, and *group signatures [ACJT00]* (escrow schemes). In the latter, a group member signs messages anonymously on behalf of the group.

6.4.1 Blind Signatures

In some applications such as e-voting systems and cryptocurrency, the signer and the author of a message are different parties, and the author does not want the signer to see the content of his message. Blind signatures [Cha83] allow the content of a message to be encrypted (blinded) before it is signed. The resulting signature can then be publicly verified against the original message.

6.4.2 Ring Signatures

Introduced by Rivest, Shamir, and Tauman [RST01], a *ring signature* can provide an anonymous signature from a member of a group without revealing which member signed the message. This is useful for whistleblowing, as hinted by the title *How to leak a secret* of [RST01]. A group of parties each have public/private key pairs $(PK_1, SK_1), (PK_2, SK_2), \ldots, (PK_n, SK_n)$. Party i can compute a ring signature σ on a message m, on input $(m, SK_i, PK_1, \ldots, PK_n)$. Anyone can check the validity of the signature given σ, m, and $PK_1, \ldots, PK_n$. The anonymity property is that any verifier should not have a probability greater than $1/n$ to guess the identity of the real signer who has computed a ring signature.

Linkable Ring Signatures. Introduced by Liu et al. [LWW04], an LRS scheme allows anyone to determine whether two ring signatures are signed by the same group member without disclosing the identity of the member. This allows a verifier to check distinctness and prevent the signer from signing or authenticating himself more than once and thus preventing Sybil attacks [Dou02] and double-voting in e-voting systems. The algorithm of [LWW04] makes $O(n)$-long signatures, where n is the group size. Au et al. [ALSY06] propose a constant-size LRS algorithm.

7 Interactive Proofs

An *interactive proof* is a conversation between a prover and a verifier that ends with the verifier either accepting or rejecting the prover's statement. The prover and the verifier exchange *challenges* and *responses*, typically dependent on random numbers that they are allowed to keep secret. Interactive proofs used for identification may be formulated as proofs of knowledge. In this scenario, Alice has a secret s, and she wishes to convince Bob that he has knowledge of s by responding correctly to queries made by Bob that require knowledge of s to answer.

An interactive proof is called to be a *proof of knowledge (PoK)* if it provides two properties: *soundness* and *completeness:*

- **Completeness:** An interactive proof protocol is *complete* if, given an honest prover and an honest verifier, the protocol succeeds with overwhelming probability (i.e., the verifier accepts the prover's claim).

Informally, completeness means that the protocol is correct for honest prover and honest verifier.

- **Soundness:** An interactive proof protocol is *sound* if there exists an expected polynomial time algorithm that if a dishonest prover can with non-negligible probability execute the protocol with the verifier, then the algorithm can be used to extract the knowledge from the prover.

Informally, soundness means the protocol is secure against a dishonest prover. In other words, a malicious prover cannot convince an honest verifier to accept the prover's statement unless the prover knows some secret.

Σ-protocol. PoK protocols often consist of three rounds: commitment, challenge, and response. Such a protocol is called a Σ-protocol **[?].**

7.1 Zero-Knowledge Proofs

First introduced by Goldwasser, Micali, and Rackoff [?], a zero-knowledge protocol allows PoK without revealing the an information about the knowledge to a (polynomial-time) adversary. To define the zero-knowledge property, we need to first define two concepts:

View. A view (or transcript) of a PoK is the sequence of steps taken between the prover and the verifier in the PoK.

Simulator. A simulator is a procedure that can generate fake views that are indistinguishable from the original view of the PoK generated with the prover.

Zero-Knowledge Proof. A zero-knowledge proof (ZKP) is a proof of knowledge that can additionally provide the following property:

- **Zero-Knowledge:** A PoK has the zero-knowedge property if there exists a simulator for the proof.

Informally, this means that no amount of knowledge should pass between the prover and the verifier that the verifier could not figure out without the help of the prover. In other words, if an adversary can successfully verify the proof after communicating with the prover, then the adversary could just as well have succeeded without the prover by just running the simulator. This guarantees that the number of times that the prover participates in them does not vary the chances of success of impersonation attacks.

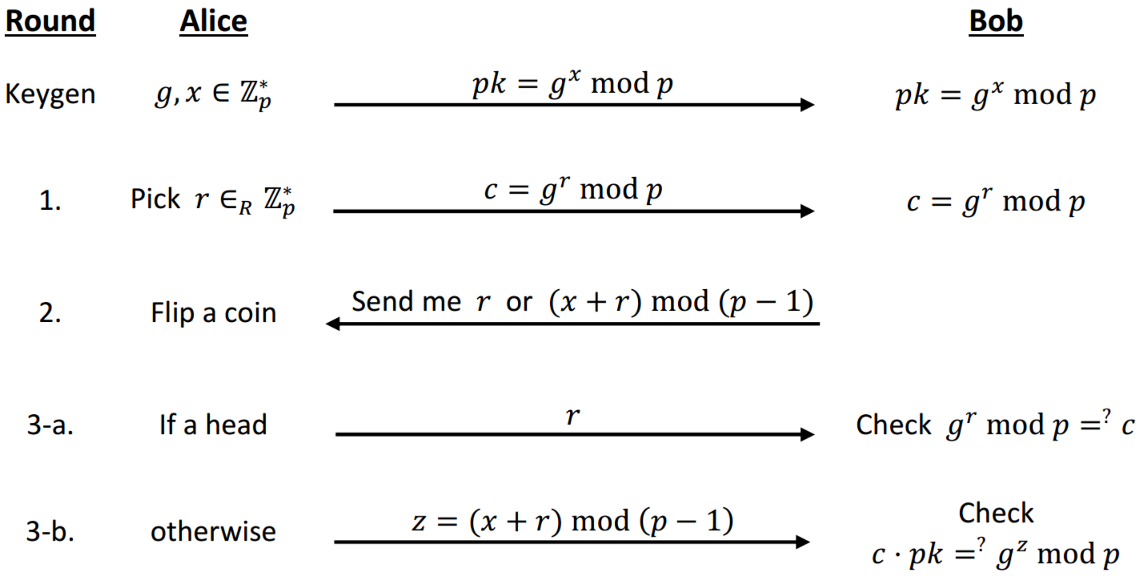

Round
Alice
Bob
Keygen
$g, x \in \mathbb{Z}_p^*$
$pk = g^x \bmod p$
$pk = g^x \bmod p$
1.
Pick $r \in_R \mathbb{Z}_p^*$
$c = g^r \bmod p$
$c = g^r \bmod p$
2.
Flip a coin
Send me r or $(x + r) \bmod (p - 1)$
3-a.
If a head
r
Check $g^r \bmod p =^? c$
3-b.
otherwise
$z = (x + r) \bmod (p - 1)$
Check
$c \cdot pk =^? g^z \bmod p$

Figure 6: Proof of knowledge of discrete log [CEG88].

7.1.1 Non-Interactive Zero-Knowledge Proof

If the proof is done with no interaction between the prover and the verifier, then it is called a *non-interactive zero-knowledge proof (NIZK)*. Goldreich and Oren [?] show that NIZK is impossible in the standard model meaning that some assumption like a common reference string (CRS) or a random oracle are needed. Using the Fiat-Shamir heuristic [?], one can construct a NIZK protocol in the random oracle model from an interactive ZK protocol.

zk-SNARK. A *zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK)* allows a constant-size ZKP in terms of the size of the circuit representing the program that its correct execution is being verified. The program is compiled into an equation of polynomials $t(x) \cdot h(x) = w(x) \cdot v(x)$, where the equality holds if and only if the program is computed correctly. The prover wants to convince the verifier that this equality holds. To reduce the proof size and the verification time, the verifier chooses a secret evaluation point s to reduce the problem from multiplying polynomials and verifying their equality to multiplication and equality check on numbers: $t(s) \cdot h(s) = w(s) \cdot v(s)$. A (partially) homomorphic encryption function E is used so that the prover can compute $E(t(s)), E(h(s)), E(w(s)), E(v(s))$ from $E(s)$ without knowing s. A zk-SNARK requires a trusted setup which generates a common reference string (CRS) for the NIZK. In this setup, the verifier chooses a random and secret field element s and encrypts the values of the polynomials at that point. For a more detailed description of a zk-SNARK see [?].

7.2 Proof of Discrete Logarithm

Chaum et al. [CEG88] propose a protocol for proving the knowledge of a discrete logarithm. Figure 6 shows the protocol rounds. In this protocol, Alice (the prover) wants to prove to Bob (the verifier) that she knows the discrete logarithm of $g^x \bmod p$, which is x.

8 Elliptic Curve Cryptography

The goal of Elliptic Curve Cryptography (ECC) is to provide the same level of security as RSA or discrete logarithm systems but with significantly shorter keys (about 160-256 bit vs. 1024-3072 bit). ECC is based on the discrete logarithm problem defined on elliptic curve groups. There is no known efficient discrete-logarithm solving algorithm for elliptic curves, beyond the generic algorithms, which work on every group.

Elliptic Curve. An elliptic curve is a set of points (x, y) defined by any equation in the form

$$y^2 = x^3 + ax + b.$$

In cryptography, such a curve is defined over finite fields rather than real numbers, i.e., for a prime p an elliptic curve over $\mathbb{Z}_p$ is defined using

$$y^2 \equiv x^3 + ax + b \bmod p,$$

where $a, b \in \mathbb{Z}_p$. This curve must be *nonsingular* meaning that it must not intersect itself. Algebraically, this means that the discriminant $\Delta = -16(4a^3 + 27b^2)$ is nonzero.

The points on an elliptic curve and a *point at infinity* form a group under a certain addition law. If P and Q are two points on the curve, then the addition law defines a unique third point, $P + Q$. A *primitive point* G is a *generator* of this group: all elements of the group can be generated as $G + G + \cdots + G$ (k times) for some k.

8.1 Elliptic Curve Diffie-Hellman

To generate a pair of private/public keys over an elliptic curve such that finding the private key from the public key is as hard as solving the discrete-logarithm problem, one chooses a pseudorandom scalar value x as his private key using a PRG. Then, he sets $x \times G$ as his public key, where G is a generator of an elliptic curve, and $\times$ denotes elliptic curve point multiplication by a scalar.

Let $(a, a \times G)$ and $(b, b \times G)$ be Alice and Bob's public/private key pairs. Alice sends $a \times G$ to Bob, and Bob sends $b \times G$ to Alice. Alice computes the shared secret using $a \times (b \times G)$ and Bob computes $b \times (a \times G)$. Since the $a \times (b \times G) = b \times (a \times G)$, the parties have agreed on the same secret.

8.2 Elliptic Curve Digital Signature

Let G be a generator of an elliptic curve with large prime order n. Also, let (x, Q_A) be Alice's private key/public key such that $Q_A = x \times G$. To sign a message m using her private key x, Alice performs the following steps:

1. Calculate $e = H(m)$, where H is a cryptographic hash function such as SHA-2;

2. Choose a random integer k from $[1, n - 1]$;

3. Calculate the point $(x_1, y_1) = k \times G$ on the elliptic curve;

4. Calculate $r = x_1 \mod n$. If $r = 0$, then go to step 2;

5. Calculate $s = k^{-1}(e + r \cdot x) \mod n$. If $s = 0$, then go to step 2;

6. The signature is (r, s).

To verify a signature, Bob performs the following steps:

1. Verify that r and s are integers in $[1, n-1]$;
2. Calculate $e = H(m)$;
3. Calculate $w = s^{-1} \mod n$;
4. Calculate $u_1 = zw \mod n$ and $u_2 = rw \mod n$;
5. Calculate the point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$;
6. The signature is valid if and only if $r = x_1 \mod n$.
7. $a \times H_3(h) - a \times H_3(h)$

9 Homomorphic Encryption

Homomorphic encryption is a form of encryption, which allows specific types of computations to be carried out on ciphertext and obtain an encrypted result which decrypted matches the result of operations performed on the plaintext. More formally, in homomorphic encryption we define a function f such that $f(x) = d(f(e(x)))$, where x is the plaintext, $e(x)$ is the encryption function and $d(x)$ is the decryption function. We call f a homomorphic function. Although a cryptosystem which is unintentionally homomorphic can be subject to attacks, if treated carefully, homomorphism can be used to perform secure computations.

An encryption scheme is called *additive homomorphic* if it is possible to compute an encryption of (m_1+m_2) from the encryptions of m_1 and m_2. This operation is called homomorphic addition and is denoted by $+_h : E(m_1 + m_2) = E(m_1) +_h E(m_2)$. Different cryptosystems support different levels of homomorphism. *Partially homomorphic encryption (PHE)* methods allow homomorphic encryption of either addition or multiplication on plaintexts. Table 1 lists various PHE methods.

Table 1: Homomorphic properties of various cryptosystems

Cryptosystem	**Homomorphic operations**
RSA [RSA78]	Multiplication
Goldwasser-Micali [GM82]	XOR
ElGamal [Elg85]	Multiplication, Exponentiation by constant
Benaloh [Ben94]	Addition
Paillier [Pai99]	Addition, Multiplication by constant

9.1 Fully Homomorphic Encryption

A cryptosystem which supports both addition and multiplication is called *fully homomorphic encryption (FHE)*. Such a system also preserves the useful algebraic properties of a ring. Using an FHE scheme, any circuit can be homomorphically evaluated. There are several efficient PHE methods but research on efficient FHE schemes is still ongoing.

An FHE scheme allows to perform *non-interactive secure computation*, which is very useful in the applications that permit small number of interactions between parties. The first FHE scheme was proposed by Craig Gentry in 2009 [Gen09], which is based on lattices and is very computationally intensive. However, in the past four years the efficiency of lattice-based FHE schemes has been improved by several orders of magnitude giving some hope to the community. The ciphertexts in all these schemes are *noisy*, with a noise that grows slightly during homomorphic addition, and explosively during homomorphic multiplication.

Somewhat Homomorphic Encryption (SHE). Current FHE schemes can only evaluate small circuits due to some noise term. In other words, they allow homomorphic operations to be performed only a limited number of times because the error term increases with each operation. Once the error exceeds a certain tolerance, the result can no longer be decrypted.

Bootstrapping. Gentry [Gen09] starts with an SHE scheme and then shows how to make it bootstrappable. An SHE scheme is *bootstrappable* if it can decrypt messages with a sufficiently small error constant. In other words, it implements a `Recrypt` function which *refreshes* the ciphertext by homomorphically decrypting and then re-encrypting the message, which reduces the error term. Using the bootstrapping procedure, we can perform unlimited number of homomorphic operations and hence, we get an FHE scheme from an SHE scheme.

10 Pairing-Based Cryptography

Let G_1 and G_2 be additive groups and G_T be a multiplicative group, all of prime order p. Also, let $g_1 \in G_1$ and $g_2 \in G_2$ be generators of G_1 and G_2 respectively. We define a *bilinear pairing* (or a *bilinear map*) to be the map $e : G_1 \times G_2 \to G_T$ such that

$$e(ag_1, bg_2) = e(g_1, g_2)^{ab},$$

for all $a, b \in \mathbb{Z}_p$ and $e(g_1, g_2) \neq 1$. Clearly, if G_1 and G_2 are multiplicative groups, then the bilinearity property becomes

$$e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}.$$

In practice, the pairing e is required to be efficiently computable. In some applications, the pairing is also required to be hard to invert.

10.1 The BLS Signature Scheme

Boneh, Lynn, and Sacham [BLS01] propose a pairing-based digital signature scheme that is secure in the random-oracle model. The scheme generates short signatures that are elements of an elliptic curve group and uses the Weil bilinear pairing to verify the signatures.

Key Generation. All parties agree on three groups G_1, G_2, and G_T of prime order p and a generator g of G_2. Each party chooses a random integer $x \in [0, p-1]$ as its private key and broadcasts g^x as the public key.

Signing. To sign a message m, first take a hash of the message $h = H(m) \in G_1$, and then compute the signature $h^x \in G_1$.

Verification. Let $e : G_1 \times G_2 \to G_T$ be a bilinear pairing. Given the signature h^x, the public key g^x, and the message m, verify that $e(h^x, g) = e(H(m), g^x)$.

Correctness can be simply proved using the bilinearity property $e(h^x, g) = e(h, g^x)$.

11 Identity-Based Encryption

In public-key encryption, if a party P wants to send encrypted messages $m_1, ..., m_n$ to $P_1, ..., P_n$, he needs to know public keys $PK_1, ..., PK_n$ of the recipients so that he sends $c_i = \mathsf{Enc}(PK_i, m_i)$ to P_i who can decrypt $m_i = \mathsf{Dec}(SK_i, c_i)$ with his own secret key SK_i. Due to the nature of the encryption scheme, public keys are usually long and obtaining them adds a communication overhead. In practice, each recipient often holds an identity that can usually be represented with a much shorter bit string and is independent of the encryption scheme.

In *identity-based encryption (IBE)*, the key generation protocol only generates a single public key that can be used along with the identity of each recipient to encrypt the corresponding recipient's message. In other words, party P encrypts m_i for party P_i using $c_i = \mathsf{Enc}(PK, ID_i, m_i)$, where PK is the common public key and ID_i is the identity of P_i.

The key generation protocol also generates a master secret key MSK which is used by each recipient P_i along with the recipient's identity ID_i to generate P_i's own secret key using $SK_i = \mathsf{Extract}(MSK, ID_i)$. Using this secret key, P_i decrypts his cipher using $m_i = \mathsf{Dec}(SK_i, c_i)$. Therefore, an IBE scheme provides four functionalities: Setup, Extract, Enc, and Dec. The most well-known IBE scheme is due to Boneh and Franklin [BF01] which is based on bilinear maps and is shown secure under bilinear Diffie-Hellman (BDH) assumption.

12 Distributed Consensus

One of the most fundamental problems in distributed computing happens when a group of parties want to agree on a common value without the help of any trusted third party while some of the parties are being controlled by an adversary. When the faulty behavior is limited to crash failures (a.k.a., fail-stop) but corrupt parties otherwise follow the protocol, this problem is known as *distributed consensus.* If the parties can deviate arbitrary from the protocol (i.e., are Byzantine), then this problem is known as *Byzantine agreement.*

The impossibility result of [FLP85] shows that deterministic asynchronous consensus protocols cannot guarantee protocol termination. Therefore, in many cases, we rely on probabilistic (randomized) protocols to achieve agreement with all but negligible probability. In the following section, we describe the Byzantine agreement problem and state known important results.

12.1 Byzantine Agreement

In 1982, Lamport, Shostak, and Pease [LSP82] were the first to introduce the problem of Byzantine Generals as follows:

> "...a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement."

Problem Statement. Consider n parties, of which up to t may be corrupted by an adversary and exhibit arbitrary faults; the remaining parties are honest. Every party starts out with an initial value and the goal is to agree on a common value that is equal to some honest party's input. For example, if all parties start with 1, they must all decide on 1.

Let t be the number of corrupted parties. The following important results about Byzantine agreement (BA) exists in the literature:

- For $t < n/3$, BA is possible using a deterministic protocol with round complexity $O(t)$ [PSL80] and using a probabilistic protocol with expected round complexity $O(1)$ [FM88].
- For $t < n/2$, BA is possible with cryptographic assumptions, e.g., a public-key infrastructure (PKI) [PSL80].

With the assumption of private channels, Feldman and Micali [FM85] propose a randomized BA protocol with constant expected time. Later, Canetti and Rabin [**?**] achieved the optimal resiliency bound of $t < n/3$ for asynchronous BA with private channels. With cryptographic assumptions, Cachin et al. [CKS00] achieve constant communication and polynomial computation complexities in the asynchronous model with adaptive adversary.

In the full information model, Kapron et al. [**?**] propose a polylogarithmic BA assuming the adversary is static. Their protocol elects a logarithmic subset of parties less than 1/3 of which are corrupted with high probability. When the adversary is adaptive, the best known BA protocol in the full information model has expected polynomial communication and computation complexities [KS16].

12.2 Related Problems

Gossiping Protocol. In a basic gossip protocol, there is a fixed group of participants, where each participant randomly selects a peer to exchange its state. This state is updated in such a way that all honest participants eventually converge to the same state. It can be shown that this approach is efficient, converging in $O(\log n)$ gossip rounds where n is the number of participants, even in the face of participants failures and message loss. There are gossip protocols that allow open and dynamic membership, where the membership itself is gossiped along with other state.

Reliable Broadcast. A broadcast channel (a.k.a., reliable broadcast channel or simply, broadcast) allows one party to send a message to all other parties, guaranteeing consistency even if the broadcaster is dishonest. This problem can be equivalently seen as a Byzantine Agreement (BA) problem, where the parties want to agree on a given message. Hence, a broadcast channel can be simulated via BA. A broadcast channel cannot be simulated on a point-to-point network when a 1/3 or more of the parties are dishonest because BA is impossible in such a setting [LSP82]. A broadcast channel is usually viewed as an expensive resource since BA is expensive[6]. For this reason, many cryptographic protocols, which assume the existence of broadcast channel, report their costs as the number of times a secure broadcast is done and try to minimize this number. A consensus protocol, where each party provides a vote to agree on one of the votes usually requires multiple invocations of a secure broadcast protocol.

Leader Election. Some good processor p is elected as the leader and is known by all processors. With constant probability, p is good.

Universe Reduction. This is a generalization of *Leader Election.* Some set C of size $O(\log^3 n)$ is elected and known by all processors. With high probability $(1 - o(1))$, C is good i.e. a majority of the processors in C are good.

[6] Byzantine agreement is expensive in terms of both communication and round complexity.

We can solve the above three problems above with each processor sending $\tilde{O}(\sqrt{n})$ bits using the protocol of King et al. [KS09].

Almost-Everywhere Agreement. It is a relaxed version of the BA problem, which was first introduced by Dwork et al. in [DPPU86]. Instead of bringing all good processors to agreement, a large majority of good processors are brought to agreement. This problem was first solved efficiently by [**?**] with poly-logarithmic message, computation, and round complexity for each processor.

Almost-Everywhere to Everywhere. It is straightforward to go from everywhere UR to everywhere agreement for BA and LE. Any *representative subset* of processors run a standard BA or LE protocol and then, the representative set communicates its results to the other processors so the problem is solved for the whole set of processors.

Almost-Everywhere Universal Reduction. The almost-everywhere protocol is based on the scalable leader election protocol proposed in [**?**]. The protocol uses a layered network. Every processor is assigned to a specific set of nodes of logarithmic size on layer 0 of the network using a sampler. In order to assign processors to a node A on layer $l > 0$, the set of processors assigned to nodes on layer $l - 1$ that are connected to A hold an election. Since bad processors may be inconsistent in the election algorithm, the edges between the layers of the network are determined by samplers to disperse possible coalitions of bad processors to reduce the probability of having bad processors take over a node.

13 Secret Sharing

A player, called the *dealer*, wants to distribute a secret amongst a group of participants, each of whom is allocated a share of the secret. The secret can be reconstructed only when a sufficient number of shares are combined together and individual shares are of no use on their own, i.e. each of the shares reveals nothing to the player possessing it. Secret sharing problem generally consists of the following two phases:

1. **Sharing.** Given a private secret, build n shares of the unique secret and send each share to the corresponding player.

2. **Reconstruction (recombination).** Given the shares, build the unique secret.

There are various algorithms for sharing a secret among a group of players. Shamir's secret sharing [Sha79] is a famous one, in which any t out of n shares may be used to recover the secret. Such a method is called a *(t, n)-threshold* scheme. Note that Shamir's method does not verify that the shares sent by the dealer are consistent, i.e., they are on a valid polynomial representing a secret. Shamir's secret sharing is information-theoretically secure only over finite fields. This is because there is no uniform distribution over infinite fields, so any choice of random numbers necessarily leaks some information about the secret.

13.1 Verifiable Secret Sharing

In the standard secret sharing, players may be dishonest but the dealer must be honest. In verifiable secret sharing, the dealer may also be dishonest and distribute invalid shares among the players. Invalid (inconsistent) shares result in a secret that can never be reconstructed. If we use a method to ensure the dealer sends shares of a real secret and not just some random numbers, then we call the new scheme *Verifiable Secret Sharing (VSS)*. The problem of VSS is to convince shareholders that their shares (collectively) are, *t-Consistent*, meaning that every subset of t shares out of n defines the same secret. In such a scheme, the players *commit* to the shares they send so that they cannot change it later. In cryptography, there are certain *commitment schemes* that address this problem (see section 6).

There are two types of VSS methods: *interactive* and *non-interactive*. Interactive methods send more messages and all known non-interactive methods make hardness assumptions (i.e., are not unconditionally-secure). VSS proposed in [BGW88] is unconditionally secure and interactive. [Fel87] proposes a non-interactive VSS that is quite efficient but it has computational assumptions (intractability of computing discrete logarithm). The VSS proposed in [Ped91a] is non-interactive and is only unconditionally-secure for the dealer (i.e. dealer is unconditionally-secure against faulty parties. The dealer is kind of trusted).

The VSS of [Fel87] is based on Shamir's secret sharing scheme and a homomorphic encryption paradigm. First, the secret is shared using a random polynomial P as defined in the Shamir's sharing phase. To make the shares verifiable, the dealer also distributes commitments to the coefficients of P. Then, any party can verify the shares. The reconstruction is performed using homomorphism. As an example, suppose that we want to share a secret x and the goal is to compute a function $f(x)$. We can define the function f in a way such that $f(x) = c(f(s(x))$, where x is the secret, $s(x)$ is the sharing function, and $c(x)$ is the construction function. First, each share s_i from $s(x) = (s_1, s_2, ..., s_n)$ is sent to the corresponding player. Then, in the construction phase, each player broadcasts $f(s_i)$. Finally, the players build $f(x)$ using $f(x) = c((f(s_1), f(s_2), ..., f(s_n)))$.

13.1.1 Publicly Verifiable Secret Sharing

A VSS scheme allows honest participants to ensure that they can recover a unique secret. In contrast, a PVSS scheme allows anyone (not only the honest participants) to verify the consistency of the shares distributed by a dealer in a secret sharing scheme. Apart from the applications for ordinary VSS, PVSS can be used for new escrow-cryptosystems, and for the realization of digital payment systems with revocable anonymity. Consider n participants with a

1. The dealer chooses a degree $t - 1$ polynomial $s(x) = \sum_{j=0}^{t-1} \alpha_j^j$ and, for each paticipant i, creates an encrypted share $\hat{S}_i = X$

13.2 Secret Reconstruction

For $d \geq 0$, let $a_0, ..., a_d$ be $d + 1$ distinct elements in a finite field $\mathbb{F}$, and let $b_0, ..., b_d$ be $d + 1$ arbitrary elements of $\mathbb{F}$. Then, there exists exactly one polynomial $f(x)$ of degree less than or equal to d such that $f(a_i) = b_i$ for $i = 0, ..., d$. This polynomial is given by

$$f(x) = \sum_{i=0}^{d} b_i \prod_{j=0, j \neq i}^{d} (a_i - a_j)^{-1}(x - a_j), \tag{1}$$

where $u - v$ is an abbreviation of $u + (-v)$ with $-v$ being the additive inverse of v in $\mathbb{F}$, and u^{-1} is the multiplicative inverse of u in $\mathbb{F}$. This formula is applicable over any field and is based on Lagrange polynomials for interpolation of polynomials [HR10]. If none of the parties send invalid shares, then they can reconstruct the correct secret if less than a 1/2 fraction of the parties fail to send their shares, e.g., due to a stop and fail fault.

Theorem 1. *Given a set of $d + 1$ shares $\{s_i\}$, Protocol 1 correctly reconstructs the corresponding secret.*

Proof. Using Shamir's scheme [Sha79], the constant term in polynomial of equation (1) is the secret [HR10]. Since equation (2) calculates coefficients of the constant term, equation (3) calculates the secret. □

Protocol 1 Reconstructing secret S (by player P_k)

Input: a set of at least $d+1$ shares $\{s_i\}$ each of which is received from player with index a_i, where $i = 0, ..., d$. We assume $a_0 = k$ meaning that s_0 is P_k's share. d is the degree of the random polynomial used by the dealer to create s_i's.

Output: secret S.

Calculate Lagrange's free coefficients defined by

$$L_i = \prod_{j=0, j\neq i}^{d} (a_i - a_j)^{-1}(-a_j) \tag{2}$$

Then, secret S can be reconstructed by multiplying each share s_i to the corresponding Lagrange's free coefficient and then, summing up the results, i.e.,

$$S = \sum_{i=0}^{d} L_i s_i \tag{3}$$

Let $\rho : \mathbb{F}^{d+1} \to \mathbb{F}$ be a function such that $\rho(s) = S$, where s is a vector of shares received by any player and S is the secret. We call ρ the *reconstruction function*, which can be simply defined using (3):

$$\rho(s) = \sum_{i=0}^{r} L[i] \cdot s[i] \tag{4}$$

where $L[i]$ is the i-th Lagrange's free coefficient defined in (2).

13.2.1 Malicious Case

In the malicious case, it is possible that dishonest parties send spurious shares, i.e., values that are different from values given to them in the sharing phase. In that case, an error correcting method must be used to fix the errors. Since, Shamir's secret sharing generates Reed-Solomon Codes [RS60], we can use any Reed-Solomon error correction algorithm (called *decoder*) like Berlekamp-Welch algorithm to fix the errors.

Reed-Solomon Codes. If we have n points e of which are corrupted (noisy), we can correctly find a polynomial of degree at most d that passes through the $n - e$ correct points if and only if $e < \frac{n-d}{2}$.

Berlekamp-Welch Error Correction. Berlekamp and Welch [BW86] constructed a polynomial-time algorithm for correcting errors in Reed-Solomon codes. The algorithm has a running time of $O(n^3)$, where n is the total number of points. Let $\mathbb{F}_p$ denote a finite field of prime order p, and $S = \{(x_1, y_1) \mid x_i, y_i \in \mathbb{F}_p\}_{i=1}^{n}$ be a set of n points, where $n-\varepsilon$ of them are on a polynomial $y = P(x)$ of degree t, and the rest $\varepsilon < (n-t+1)/2$ points are erroneous. Given the set of points S, the goal is to find the polynomial $P(x)$. The algorithm proceeds as follows. Consider two polynomials $E(x) = e_0 + e_1x + ... + e_\varepsilon x^\varepsilon$ of degree ε, and $Q(x) = q_0 + q_1x + ... + q_kx^k$ of degree $k \leq \varepsilon + t - 1$ such that $y_iE(x_i) = Q(x_i)$ for all $i \in [n]$. This defines a system of n linear equations with $\varepsilon + k = n$ variables $e_0, ..., e_\varepsilon, q_0, ..., q_k$ that can be solved efficiently using Gaussian elimination technique to get the coefficients of $E(x)$ and $Q(x)$. Finally, calculate $P(x) = Q(x)/E(x)$.

Note on Shamir's Secret Sharing. Assume that the dealer sends n shares of a secret to n players. Let t be the bound on the number of faulty players and k be the number of shares each player needs in order to properly reconstruct the secret. The polynomial for interpolation must be of degree $k-1$ because with k points we can interpolate a polynomial of degree at most $k-1$. k must be greater than t because otherwise faulty players can fool good players by sending shares of a spurious secret. Also, k must be less than $n-t$ because otherwise faulty players can remain silent and good players will not be able to interpolate a polynomial of degree $n-t$ or more with only $n-t$ points. Hence, $t < k < n-t$ and the smallest k is $t+1$ and he only needs to ask for $2t$ shares (or $2t+1$ including himself), t of which may never be received correctly.

14 Secure Multi-Party Computation

In *secure multi-party computation (MPC)*, a set of parties, each having a secret value (input), want to compute a common function over their inputs, without revealing any information about their inputs other than what is revealed by the output of the function. This problem was first described by Yao [Yao82]. He described an algorithm for MPC with two parties in the presence of a semi-honest adversary. Goldreich et al. [GMW87] propose the first MPC protocol that is secure against a Byzantine adversary. This work along with [CDG88, GHY88] are all based on cryptographic hardness assumptions and are often regarded as the first generic solutions to MPC. These were followed by several cryptographic improvements [BMR90, **?**, CFGN96] as well as information theoretically-secure protocols [BGW88, **?**, Bea91, BCG93] in late 1980s and 1990s. For a complete anotated bibliography of MPC results see this.

14.1 MPC Problem

In the MPC problem, a set of n parties, each holding a private input, jointly evaluate a function f over their inputs while ensuring,

1. Upon termination of the protocol, all parties have learned the correct output of f; and
2. No party learns any information about other parties' inputs other than what is revealed from the output.

We assume the identities of the n parties are common knowledge, and there is a private and authenticated communication channel between every pair of parties. We consider two communication models. In the *synchronous* model, there is an upper bound, known to all parties, on the length of time that a message can take to be sent through a channel. In the *asynchronous* model, there is no such upper bound. The standard definition of secure computation assumes a rushing adversary.

Usually, a certain fraction of the parties are controlled by a *Byzantine*[7] adversary. These parties can deviate arbitrarily from the protocol. In particular, they can send incorrect messages, stop sending any messages, share information amongst themselves, and so forth. Their goal is to thwart the protocol by either obtaining information about the private inputs, or causing the output of the function to be computed incorrectly. We say the adversary is *semi-honest* if the adversary-controlled parties are curious to learn about other parties' secret information, but they strictly follow the protocol. We say that the parties controlled by the adversary are *malicious* (or *Byzantine* or *dishonest*). The remaining parties are called *semi-honest* (or simply, *honest*).

The adversary is either *computationally-bounded* or *computationally-unbounded*. The former is typically limited to only *probabilistic polynomial-time (PPT)* algorithms, and the latter has no computational

[7] Also known as *active* or *malicious*.

limitations. The adversary is either assumed to be *static* or *adaptive*. A static adversary is limited to selecting the set of dishonest parties at the start of the protocol, while an adaptive adversary does not have this limitation.

Security Properties. The following security properties are often considered in the design of MPC protocols [GL02]:

- **Privacy**: No party should learn anything other than what is revealed from the output.
- **Correctness**: The outputs received by the parties are guaranteed to be correct.
- **Output Delivery**: Corrupted parties should not be able to prevent honest parties from receiving their output. This is sometimes called the termination property. When termination cannot be guaranteed, if any party leaves the protocol, the protocol aborts.
- **Fairness**: Corrupted parties should receive output only if honest parties do. Without this property, the adversary can see the output and then decide whether to let the honest parties receive their output or not.

MPC and Byzantine Agreement. Byzantine agreement (BA) is a special case of MPC. Thus, all lower bounds for BA immediately apply to MPC. For example, BA is impossible for any $t \geq n/3$ [PSL80], therefore general MPC is also impossible in this setting with guaranteed output delivery [GL02]. MPC protocols that work with a dishonest majority such as [DO10] cannot guarantee output delivery (termination) and fairness.

Note on Non-Deterministic Functionalities. Functionality f is often assumed to be deterministic since otherwise running multiple instances of the MPC protocol with the same inputs can reveal significant amounts of information about the inputs other than what is allowed to be revealed by the output in a single run. For example, consider a functionality $f(x)$ where a fresh random value r is generated by f in each instance and x is compared against r. If $x < r$, then f returns 0. Otherwise, it returns 1. While computing $f(x)$ for a fixed x can only reveal a small (negligible) information about x, multiple computation of f over x can reveal large information about x.

14.2 MPC Approaches

Depending on the model of computation, every function can be represented in terms of some *elementary operations* such as arithmetic operations (e.g., addition, multiplication), Boolean operations (e.g., and, or), RAM instructions (e.g., get-value, set-value), etc. Informally speaking, every MPC protocol specifies how a group of elementary operations can be computed securely. The function is computed securely via composition of these secure operations. From this perspective, we classify the broad range of MPC approaches into two categories: techniques that evaluate circuits (Boolean or arithmetic), and techniques that evaluate RAM programs.

14.2.1 Circuit-Based Techniques

We subdivide the set of circuit-based methods into three categories based on their main approach for achieving privacy: garbled circuits, secret sharing, and fully homomorphic encryption.

Garbled Circuits. The idea of garbled circuits dates back to the two-party MPC proposed by Yao [Yao82].[8] One party is called the *circuit generator* and the other one is called the *circuit evaluator*. For each wire in the circuit, the generator creates a mapping that maps each possible value of that wire to another value (called the *garbled value*). The generator then sends this mapping to the evaluator. The evaluator evaluates the circuit using the mapping to compute the *garbled output*. Next, the generator computes another mapping (called *translation*) that maps all possible garbled outputs to their actual values. In the final round, the generator sends the translation to the evaluator, and the evaluator sends the garbled output to the generator. Both parties can compute the actual output at the same time without learning anything about each other's inputs. This algorithm is only secure in the semi-honest setting. Yao's original model has been the basis for several secure computation algorithms mostly for the two-party setting with computational hardness assumptions [GMW87, LP07, HEKM11, LP11, KMR11]. In a line of research, Lindell and Pinkas give the first proof of Yao's protocol [LP09] and present a two-party approach based on garbled circuits that uses the cut-and-choose technique to deal with malicious parties [LP07, LP11].

Secret Sharing. In secret sharing, one party (called the *dealer*) distributes a secret amongst a group of parties, each of whom is allocated a *share* of the secret. Each share reveals nothing about the secret to the party possessing it, and the secret can only be reconstructed when a sufficient number of shares are combined together. Many MPC schemes build upon the notion of secret sharing (most notably, [BGW88, **?**, Bea91, **?**, DIK$^+$08, DPSZ12, DKMS14]). Informally speaking, each party secret shares its input among all parties using a secret sharing scheme such as Shamir's scheme [Sha79]. Then, all parties perform some intermediate computation on the received shares and ensure that each part now has a share of the result of the computation. In the final stage, all parties perform a final computation on the intermediate results to find the final result. In the Byzantine setting, each stage of this approach usually requires several rounds of communication used to verify consistency of the shares distributed by each party (using a *Verifiable Secret Sharing (VSS)* scheme like [CGMA85, BGW88]) and to perform complicated operations such as multiplication.

Fully Homomorphic Encryption. A *fully homomorphic encryption (FHE)* scheme allows to perform secure computation over encrypted data without decrypting it. Gentry [Gen09] proposed the first FHE scheme based on the hardness of lattice problems. Since then, many techniques have been proposed to improve the efficiency of FHE [vDGHV10, BGV12, GHS12]. Unfortunately, current techniques are still very slow and can only evaluate small circuits. This restriction is primarily due to noise management techniques (such as bootstrapping [Gen09]) used to deal with a noise term in ciphertexts that increases slightly with homomorphic addition and exponentially with homomorphic multiplication.

14.2.2 RAM-Based Techniques

Most MPC constructions model algorithms as circuits. Unfortunately, a circuit can at best model the *worst-case* running time of an algorithm because a circuit can only be created by unrolling loops to their worst-case runtime [GKP$^+$13]. Moreover, circuits incur at least a linear computation complexity in the total size of the input, while a sublinear overhead is crucial for achieving scalability in most large-scale applications. In addition, most algorithms have already been described in terms of instructions (programs) to a *random access memory (RAM)* machine[9] [CR72], not circuits. These all bring the following question to the mind: Is it possible to securely evaluate RAM programs instead of circuits? Luckily, the answer is "yes". Goldreich

[8] The term "garbled circuits" is due to Beaver, Micali, and Rogaway [BMR90].

[9] A RAM machine has a *lookup functionality* for accessing memory locations that takes $O(1)$ operations. Given an array A of N values and an index $x \in \{1, \dots, N\}$, the lookup functionality returns $A[x]$.

and Ostrovsky [GO96] show that by constructing a RAM with secure access, one can evaluate arbitrary RAM programs privately. Such a RAM is often called an *Oblivious RAM (ORAM)*. This is typically considered in a setting, where a group of parties (clients) want to access a data storage (RAM) held by another party (a server).

15 Proof of Security Techniques

The first step in proving the security of a cryptographic protocol is to define what *secure* means. In this section, we describe three standard techniques for proving security of cryptographic algorithms.

15.1 Game-Based Proofs

The security of a cryptographic algorithm is phrased as a *game* played between an *adversary* (or *attacker*) and a hypothetical entity called the *challenger*. Both adversary and challenger are probabilistic processes that communicate with each other, and so we can model the game as a probability space. Typically, the definition of security is tied to some particular event S. In this context, security means that for every *efficient* adversary, the probability that event S occurs is *very close* to some specified *target probability*: typically, either 0, 1/2, or the probability of some event in some other game in which the same adversary is interacting with a different challenger [Sho04].

In most cases, we show that if there exists a polynomial-time adversary $\mathcal{A}_{Alg}$ that breaks the security of our algorithm with non-negligible probability, then we can use this adversary to create an adversary $\mathcal{A}_H$ that solves a known hard problem H (like the Decisional Diffie-Hellman problem) with non-negligible probability. Since $\mathcal{A}_{Alg}$ can break the security property with non-negligible probability, it has a non-negligible *advantage* ϵ_{Alg} in the security game. We then show that if $\mathcal{A}_{Alg}$ exists, then we can use $\mathcal{A}_{Alg}$ as a subroutine in $\mathcal{A}_H$ in its game with challenger C_H.

As an example, the semantic security of an encryption scheme is defined as a game between an adversary $\mathcal{A}$ and a challenger in the following way. The adversary starts the game by sending two messages M_0 and M_1 to the challenger. The challenger then generates a random key k, flips a coin $b \in \{0, 1\}$, and outputs $\mathsf{E}(k, m_b)$ to the adversary. Finally, the adversary guesses whether he was given the encryption of M_0 or the encryption of M_1 by outputting a bit $b' \in \{0, 1\}$. The semantic security advantage of the adversary $\mathcal{A}$ against the encryption scheme E is defined as

$$\mathsf{Adv}[\mathcal{A}, \mathsf{E}] = \Big|\mathsf{Pr}[b' = 1|b = 0] - \mathsf{Pr}[b' = 1|b = 1]\Big|.$$

We say the encryption scheme is semantically-secure if for all efficient adversary $\mathcal{A}$, the advantage $\mathsf{Adv}[\mathcal{A}, \mathsf{E}]$ is negligible.

15.2 Simulation Paradigm

The *simulation paradigm* (a.k.a., the *real/ideal model*) is a security proof technique first described by Goldreich *et al.* in [GMW87]. In contrast to defining a security game with one adversary in terms of a success probability, the security of a protocol is proved by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario (*simulation*), which is secure by definition. In the ideal scenario, there is an incorruptible trusted party to whom the parties send their inputs. We refer to the algorithm run by the trusted party in the ideal model as the *functionality* of the protocol. In the real model, parties run the actual protocol that assumes no trusted party. We refer to a run of the protocol in one of these models as the *execution* of the protocol in that model.

A protocol is *secure* if any adversary in the real model (where no trusted party exists) can do no more harm than if it was involved in the ideal model [Gol00, Section 4.3]. More formally, a protocol $\mathcal{P}$ securely computes a functionality $F_{\mathcal{P}}$ if for every adversary $\mathcal{A}$ in the real model, there exists an adversary $\mathcal{S}$ in the ideal model, such that the view of the adversary from a real execution of $\mathcal{P}$ with $\mathcal{A}$ is indistinguishable from the view of the adversary from an ideal execution with $\mathcal{S}$. The adversary in the ideal model, $\mathcal{S}$, is called the *simulator.*

Let I denote the set of indices of corrupted parties, VIEW_{REAL} denote the view of the adversary from the real execution, and VIEW_{IDEAL} denote the view of the adversary from the ideal execution. The simulator performs the following operations:

1. Generate dummy inputs $\{d_i\}$ for each honest party $i \notin I$ and receives the actual inputs $\{x_i\}$ of corrupted parties $i \in I$;

2. Run $\mathcal{P}$ over $\{d_i\}_{i \notin I}$ and $\{x_i\}_{i \in I}$ and add all messages sent/received by corrupted parties to VIEW_{IDEAL};

3. Send the inputs of corrupted parties $\{x_i\}_{i \in I}$ to the trusted third party;

4. Receive the outputs of corrupted users $\{y_i\}_{i \in I}$ from the trusted third party and add them to VIEW_{IDEAL}.

Meanwhile, a real instance of $\mathcal{P}$ is executed with actual inputs for all parties, and VIEW_{REAL} is created by gathering inputs of corrupted parties, messages sent/received by corrupted parties during the protocol and their final outputs. Once the simulation is finished, the security is proved by showing that VIEW_{IDEAL} is indistinguishable from VIEW_{REAL}.

15.3 Universal Composability

When a protocol is executed several times possibly concurrently with other protocols, one requires to ensure this composition preserves the security of the protocol. This is because an adversary attacking several protocols that run concurrently can cause more harm than by attacking a *stand-alone* execution, where only a single instance of one of the protocols is executed (see [Gol04] Section 7.7.2). One way to ensure this is to show the security of the protocol in the *universal composability (UC)* framework of Canetti [Can01]. A protocol that is secure in the UC framework is called *UC-secure.*

The simulation paradigm provides security only in the stand-alone model. To prove security under composition, the UC framework introduces an adversarial entity called the *environment*, denoted by $\mathcal{Z}$, who generates the inputs to all parties, reads all outputs, and interacts with the adversary in an arbitrary way throughout the computation. The environment also chooses inputs for the honest parties and gets their outputs when the protocol is finished.

A protocol is said to *UC-securely* compute an ideal functionality if for any adversary $\mathcal{A}$ that interacts with the protocol there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with a run of the protocol and $\mathcal{A}$, or with a run of the ideal model and $\mathcal{S}$. Now, consider a protocol $\mathcal{P}$ that has calls to ℓ subprotocols $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ which are already proved to be UC-secure. To facilitate the security proof of $\mathcal{P}$, we can make use of the *hybrid model*, where the subprotocols are assumed to be ideally computed by a trusted third-party. In other words, we replace each call to a subprotocol with a call to its corresponding functionality. This hybrid model is usually called the $(\mathcal{P}_1, \ldots, \mathcal{P}_\ell)$*-hybrid* model. We say $\mathcal{P}$ is *UC-secure in the hybrid model* if $\mathcal{P}$ in the hybrid model is indistinguishable by the adversary from $\mathcal{P}$ in the ideal model. The *modular composition theorem* [Can00] states that if $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ are all UC-secure, and $\mathcal{P}$ is UC-secure in the hybrid model, then $\mathcal{P}$ is UC-secure in the real model.

16 Program Obfuscation

In program obfuscation, a given program P_1 is converted to another program P_2 that represents the same functionality (i.e., the same I/O) while an adversary who can see P_2 cannot understand its logic. From a cryptographic perspective, this is like having a software that can keep a secret. This leads to a new notion of program obfuscation called *indistinguishability obfuscation (iO)*, where a polynomial-time adversary cannot distinguish between the obfuscations of two equivalent programs. The first mathematical construction of an indistinguishability obfuscator was proposed by Garg-Gentry-Halevi in 2013. The main idea to obfuscate programs using structured noise rather than just random noise. When the program is evaluated, the noise cancels out and the correct output is obtained.

17 Interactive Communication

How can two parties run a protocol over a noisy channel? The *interactive computation* addresses this question. Consider two parties connected by a communication channel. Each party has an input to a computation that can be performed using an interactive protocol Π known to both parties. This protocol is defined as a sequence of n symbols from a finite alphabet Σ and assumes the channel is noise-free. However, the channel is usually subject to some *random* or *adversarial noise* corrupting any ϵ fraction of the symbols transmitted between the parties.

In *interactive communication*[10], the goal is to *simulate* Π using another protocol (called *simulation*) Π' over a noisy channel such that Π' achieves the same outcome as Π. Let $|\Pi|$ denote the number of symbols transmitted by a protocol Π. The *communication rate* of an interactive protocol Π' is defined as $|\Pi|/|\Pi'|$ which shows how efficient Π' can simulate Π over a noisy channel. The *interactive channel capacity* is defined as the maximum communication rate that can be achieved over a noisy channel. In general, we like to achieve better communication rate, as close to the channel capacity as possible.

An interactive protocol has either an *alternating* or *non-alternating* communication order. In an alternating communication, the two parties take turns sending one symbol each. If either of them does not have a message to sent, it sends a NULL message. In a non-alternating communication, there is no such a restriction, *i.e.*, the communication can proceed in any arbitrary order.

From another perspective, an interactive protocol has either a *non-adaptive* or *adaptive* communication order. A non-adaptive protocol has *a priori* fixed communication order which defines for each time step which party sends and which listens. In the adaptive case, there is no such a restriction meaning that the protocol can determine the order of communication during its runtime.

In the case of *one-way* communication, where only Alice wants to send information to Bob, Shannon proves that the communication rate $1-H(\epsilon)$ is the exact upper and lower bound for reliable communication in the presence of random noise, where $H(\epsilon) = \epsilon \log 1/\epsilon + O(\epsilon)$ for $\epsilon \to 0$. For the adversarial noise setting, Hamming shows that $1 - \Theta(H(\epsilon))$ is possible, but the optimal rate is still an open question. In this setting, the Gilbert-Varshamov bound of $R \leq 1 - H(2\epsilon)$ seems tight.

In the *two-way (interactive)* communication setting, Schulman [Sch96] show that tolerating an $\epsilon = 1/240$ fraction of the adversarial errors is possible for some constant overhead. Kol and Raz [KR13] show a $1 - \Omega(\sqrt{H(\epsilon)})$ bound on the channel capacity for the random noise model. They assume a complex input protocol and non-adaptive simulations. Assuming arbitrary input protocols and adaptive simulations, Haeupler [Hae14] proposes a coding scheme that achieves a communication rate of $1 - O(\sqrt{\epsilon})$ exceeding the capacity bound of Kol and Raz. Haeupler [Hae14] conjectures $1 - \Theta(\sqrt{\epsilon})$ and $1 - \Theta(\sqrt{\epsilon \log\log 1/\epsilon})$ to

[10] also called *interactive computation*

be the interactive channel capacities for random and adversarial noise settings respectively.

The main difference that makes coding schemes for two-way communication harder than one-way communication is as follows. In the one way communication, Alice knows everything she wants to transfer beforehand. This allows her to send some checksums along with her messages to protect against both random and adversarial errors. Shannon showed that one needs to send at most $nH(\epsilon)$ of such checksums to ensure robust communication. In the two-way communication, Alice's transmissions depend highly on Bob's answers. If an error is detected at some point of the conversation, both parties have to *backtrack* to where the error happened.

Most previous work generate simulations that use a different alphabet than the original protocol. Haeupler [Hae14] argues that it is necessary to keep the alphabet of Π' the same as Π (why?). When the alphabet size is large, say each symbol has $\Theta(\log n)$ bits, Haeupler [Hae14] proposes a simple coding scheme using $\Theta(\log n)$-bit hash values incurring. This scheme achieves a rate of $1 - O(\sqrt{\epsilon})$ against any adversarial channel. When the alphabet is small (e.g., constant-bit symbols), designing an efficient coding scheme becomes harder because smaller (e.g., constant-sized) fingerprints are required.

Haeupler [Hae14] argues that generating adaptive simulations is necessary for simulating arbitrary protocols. If Π is adaptive and non-regular, then Π' must be adaptive unless we tolerate a constant factor in the communication rate. This is because Π needs to be simulated in order, and the time to compute a block of Π highly depends on the errors that occur during the simulation.

18 Permutation Networks

Consider n items located in n positions; we want to relocate them at random so that each position is given exactly one item and the resulting allocation has (almost) uniform probability distribution over S_n, the set of all $n!$ possible permutations of n elements. Efficient permuting at random is a challenging problem [CKLK01].

A circuit of *swapping gates* or *switches* (also called a *switching network*) may be used for generating permutations via setting its switches at random. By setting the switches uniformly at random, we obtain a certain probability distribution over S_n. We want this distribution to be *close* to the uniform distribution. There are many of so-called *permutation networks* (like [Wak68]) that give a permutation of the inputs but none of them gives a uniform distribution S_n.

Several permutation networks have been proposed consisting entirely of switches with swapping probability of $1/2$. Such networks cannot generate a uniform distribution as shown in the following argument. For a circuit with m such gates, there are 2^m equi-probable possible outcomes. Since $n!$ is not a power of 2, some of the outcomes will be redundant (see [Knu98] Section 5.3.4).

We measure the distance between two probability distributions u and v on a sample space Ω, using *variation distance*, defined as

$$d(u,v) = 1/2 \sum_{\omega\in\Omega} |u(\omega) - v(\omega)|$$

A permutation network is said to generate *almost-random permutations* if the variation distance between the probability distribution of the permutations generated by the network and the uniform distribution is less than or equal to $\frac{1}{n^c}$, for some positive constant, i.e.,

$$\sum_{\pi\in S_n} |\Pr(\pi) - \frac{1}{n!}| \leq \frac{1}{n^c}.$$

Czumaj et. al [CKLK01] propose a permutation network for generating almost-random permutations with variation distance $O(1/n^2)$. Their network has depth $O(\log^{15} n. \log\log n)$. As shown in [BGT13], the distance can be reduced by cascading r copies of the network of [CKLK01] to achieve a distance of $O((2/n)^r)$ and network depth of $O(r.\text{polylog}(n))$. In order to make the distance negligible, we can let $r = n$. However, this will result in a permutation network with $O(n)$ depth (latency), which is impractical.

Unfortunately, measuring the almost-randomness of permutations using the variation distance cannot determine whether a few permutations have large distance from $\frac{1}{n!}$ or many have small distance from $\frac{1}{n!}$. In a setting where an adversary tries to guess the permutation generated by the network, the former case gives the adversary more advantage over the latter case. So, the variation distance measure is a good choice for adversarial settings if the total distance from uniform distribution is evenly distributed over all permutations. This property can be captured by the *maximum distance*, define as

$$\max_{\pi \in S_n} |\Pr(\pi) - \frac{1}{n!}| \leq \frac{1}{n^c}.$$

Another good choice for this setting is the *squared variation distance*, defined as

$$\sum_{\pi \in S_n} |\Pr(\pi) - \frac{1}{n!}|^2 \leq \frac{1}{n^c}.$$

18.1 Oblivious Input Switching

Let $f(x_1, x_2, r)$ be the function that permutes x_1 and x_2 based on random input bit r. f can be defined as follows

$$f(x_1, x_2, r) = rx_1 + (1 - r)x_2$$

The reconstruction function $\rho(s)$ defined in section 13.2 can be used to define a new function $h(s_1, s_2, r)$, where

$$h(s_1, s_2, r) = r.\rho(s_1) + (1 - r).\rho(s_2)$$

where s_1 and s_2 are the sets of shares received by any player for x_1 and x_2 respectively. The MPC circuit equivalent to function h can be used in each permutation gate (i.e. G_A) to securely permute secret-shared inputs.

19 Sorting Networks

A *sorting network* is a network of *comparators*. Each comparator is a gate with two input wires and two output wires. When two values enter a comparator, it outputs the lower value on the top output wire, and the higher value on the bottom output wire. Sorting networks are useful for parallel execution of sorting algorithms.

Ajtai et al. [AKS83] propose an asymptotically-optimal (depth $O(\log n)$) sorting network called *AKS*. Unfortunately, the AKS network is not practical due to large constants hidden in the depth complexity. Batcher [Bat68] constructs a simple and efficient sorting network with depth $1/2 \log n(1 + \log n) = O(\log^2 n)$. The idea of Batcher's network is to sort $n = m_1 + m_2$ elements by sorting the first m_1 and the last m_2 independently, and then applying a (m_1, m_2)-*merging network* to the result. Leighton and Plaxton [LP90] propose a practical *probabilistic sorting circuit* that sorts with very high probability meaning that it sorts all but $\epsilon \cdot n!$ of the $n!$ possible input permutations, where $\epsilon = 1/2^{2^{k\sqrt{\log n}}}$, for any constant $k > 0$. For example, for $k = 2$ and $n > 64$, we get $\epsilon < 10^{-9}$.

19.1 Comparing Secret-Shared Inputs

Let $f : \mathbb{F} \rightarrow \mathbb{F}$ be the function for comparing two inputs,

$$f(x_1, x_2) = \begin{cases} (x_1, x_2), & \text{if } x_1 \geq x_2 \\ (x_2, x_1), & \text{otherwise} \end{cases}$$

The reconstruction function $\rho(s)$ defined in section 13.2 can be used to define a new function $h(s_1, s_2, r)$, where

$$h(s_1, s_2, r) = r.\rho(s_1) + (1 - r).\rho(s_2)$$

where s_1 and s_2 are the sets of shares received by any player for x_1 and x_2 respectively. The MPC circuit equivalent to function h can be used in each comparison gate (i.e. G_A) to securely compare secret-shared inputs.

20 Expander Graphs

Informally, an *expander* is a graph in which the neighborhood of any set of vertices S is large relative to the size of S. This means that every subset of vertices expands rapidly. A complete graph has the best expansion property, but in most applications sparse expander are required; ideally, the graph should have few (linear) edges, and in fact be of bounded degree, i.e., a good expander has low degree and high expansion parameters.

Expanders are useful in many applications such as error-correcting codes, pseudorandom generators, fault-tolerant algorithms, and sorting networks. The following are three main approaches for generating expanders.

- *Combinatorial:* we look at subsets of nodes in a graph and their boundaries to measure expansion. A large boundary means a large expansion.
- *Linear algebraic:* we look at the eigenvalues of a graph to find spectral gaps. A large spectral gap means a large expansion.
- *Probabilistic:* we look at random walks on a graph. If the random walks has large mixing property, then the expander has a large expansion property.

A graph $G = (V, E)$ is said to be *ϵ-expanding* if for all subsets S of V of size at most $|V|/2$, the number of edges that connect vertices of S to vertices in $V - S$ (shown with $e(S, V - S)$) is larger than $\epsilon|S|$, i.e.,

$$e(S, V - S) \geq \epsilon|S|.$$

The edge expansion of G denoted by $\phi(G)$ is defined as

$$\phi(G) = \min_{S \in V,\ |S| \leq |V|/2} \frac{e(S, V - S)}{|S|}.$$

It can be shown via the probabilistic methods that a sparse random graph is quite likely to be an expander. However, giving an explicit construction of an expander is a much harder problem because the problem of deciding whether a graph is an expander is known to be co-NP-Complete. So, the challenge is to find explicit and efficient constructions of expanders.

20.1 Random Walks on Expanders

A *random walk* on a connected undirected graph consists of the following steps: starting at a vertex v_0, we proceed at the first step to a neighbor of v_0, say v_1, chosen uniformly at random. At the second step, we proceed to a randomly chosen neighbor of v_1, and so on. The choice at each step is independent of all previous choices.

Random walk on an expander is rapidly mixing. This means that given any starting vertex, after a small number of steps, we expect the random walk to be at a uniformly distributed vertex independent of the choice of the initial vertex. In other words, the set of x vertices visited by a random walk on an expander graph is similar to a set of x vertices sampled uniformly and independently [HLW06].

20.2 Samplers

Samplers are families of bipartite graphs, which define subsets of elements such that all but a small number contain at most a fraction of "bad" elements close to the fraction of bad elements of the entire set. Samplers can be used to disperse possible coalitions of bad processors to reduce the probability of having bad processors take over a quorum.

20.2.1 Sampler construction

The edges between the layers of the network shown in Figure **??** are determined by a $(1/\log n, 1/\log n)$-sampler H_l, where $r = r_l$, $s = s_l$, and $d = \Theta(\log^8 n)$. It can be easily shown via the probabilistic method that such a sampler exists [**?**]. However, the construction is a harder task. An exact correspondence between extractors and averaging samplers is given in [Zuc97].

Definition [GVZ06]. Let $[r]$ denote the set of integers $\{1, ..., r\}$, and $[n]^d$ denote the set of all subsets of $[n]$ that consist of d elements of $[n]$. A function $H : [r] \to [n]^d$ is called a (θ, ϵ)-*sampler* if for every $T \subseteq [n]$,

$$\Pr\left[\frac{|H(s) \cap T|}{d} > \frac{|T|}{n} + \theta\right] \leq \epsilon.$$

Intuitively, a sampler is a function that, given a *random seed* $s \in [r]$, samples a set of d elements randomly from $[n]$ such that for every subset T of $[n]$, the fraction of elements in the sample set that are also in T is roughly equal to the fraction of elements of $[n]$ that are also in T. In other words, assuming T represents the set of *bad* elements in $[n]$, a sampler guarantees that the fraction of bad elements in the sampled set is bounded by $\frac{|T|}{n} + \theta$ with probability $1 - \delta$, where $\frac{|T|}{n}$ is the total fraction of bad elements in $[n]$.

When the seed s is fixed, the sampler can be represented as a bipartite graph with vertices $u_1, ..., u_n$ corresponding to the elements of $[n]$ and a vertex v corresponding to $H(s)$ such that there exists an edge between u_i and v if $i \in H(s)$.

Definition [?]. Let $G(L, R)$ be a bipartite graph with two disjoint sets of vertices L and R. Let $\Gamma(v)$ denote the set of all neighbors of vertex $v \in R$, and let $H_v = \Gamma(v)$. G is a (θ, ϵ, N)-*election graph* if the number of $v \in R$ such that H_v is *not* a (θ, δ)-sampler is less than N.

21 Bitcoin and Blockchains

Bitcoin is a digital currency introduced by Satoshi Nakamoto in 2009 [Nak08]. Bitcoin relies on cryptographic principles and a peer-to-peer network to provide secure payment transactions without relying on any trusted third party (e.g., banks, brokers, etc). This allows any two parties to transact directly with each other (for example to exchange money) without the need for a trusted third party. Primary security challenges of bitcoin are:

- **Unauthorized Spending.** Alice sends a bitcoin to Bob. Carol observing the transaction might want to spend the bitcoin Bob just received, but she cannot sign the transaction without the knowledge of Bob's private key.
- **Double-Spending.** Alice sends a bitcoin to Bob and later sends the same bitcoin to Carol. To protect against double-spending, Bitcoin relies on a distributed ledger (called the *blockchain*) that records all transactions and is visible to all parties.
- **Deanonymization of Clients.** Bitcoin clients are assigned pseudonyms that are used to identify the sender and the receiver of the money in a transaction. Although the bitcoin network does not directly store any information mapping the pseudonyms to their corresponding IP addresses, all transaction are stored publicly and thus one can conduct attacks to map transactions to their IP addresses.

Since bitcoin is a completely peer-to-peer system, it cannot rely on a trusted authority to authenticate its clients. A client can create multiple public-private key pairs (addresses) and use different key pairs in different transactions. The set of all key pairs associated with a client is called a *bitcoin wallet.* A client can transfer bitcoins between its own addresses just like transferring bitcoins to other people (i.e., via the blockchain).

21.1 Blockchain

A blockchain is a continuously-growing distributed database that can protect against tampering of its records (a.k.a., *blocks*). Each block contains a group of *transactions* that have been sent since the previous block. Satoshi Nakamoto designed the first blockchain and described it as a distributed timestamp server to prevent double-spending of money in bitcoin [Nak08].

In bitcoin, all parties hold a copy of the blockchain, which they keep updated by passing along new blocks to each other. The main challenge in implementing a blockchain on a peer-to-peer network is to prevent double-spending. Bitcoin solves this by running a decentralized voting protocol, where it incentivizes parties to *vote* for the next block to be added to the blockchain. Those parties who participate in this majority decision are called *miners*.

New transactions are broadcast to all parties. Each miner collects new transactions into a block and solves a cryptographic puzzles associated with the new block. Once he solves the puzzle, he broadcasts the new block to all parties. The miners verify the solution (via a proof-of-work; see Section 21.1.1) and all transactions in the block. If accepted, the miners work on creating the next block in the blockchain using the hash of the accepted block.

Bitcoin Incentive. The miner who finds a block is rewarded with a new (i.e., never used before) bitcoin which is transferred to him via the first transaction of the block. The block reward creates an incentive for miners to support the network and to generate bitcoins.[11]
As of 2016, mining for bitcoins is more profitable than transaction fees. As more bitcoins are mined, the reward per bitcoin will diminish toward zero and the miners will profit more from processing transactions. The block reward started at 50 bitcoins in the *genesis block*, the first block ever created in 2009. This reward

[11] Bitcoins are generated only from block rewards and transaction fees

is divided by half every 210,000 blocks which take about 4 years to be mined on average.[12] As more bitcoins are mined, the block rewards will get smaller. As a result, transaction fees will become the only source of profit for miners.

Mining Pools. When the difficulty for mining increased to the point where it could take a long time (like years) for individual miners to generate a block, bitcoin mining pools were created. With mining pools, miners can pool their resources together and share their computing power while splitting the mining reward equally according to the amount of shares they contributed to solving a block.

Transaction Processing Limit. As of 2016, Bitcoin's current block size limit is 1MB. To calculate the number of Transactions Per Second (TPS) Bitcoin can process, we divide the block size limit (1MB) by the average size of transactions (250 bytes), divided by the average number of seconds between blocks (600 sec). This results in 6.6 TPS. In practice, bitcoin can handle 1-3 TPS as of September 2016 [Blo16].

21.1.1 Proof-of-Work

Introduced by Dwork and Noar [DN93], a proof-of-work is a proof showing that some moderately hard work has been done by the prover. Such proofs should be difficult to produce, but should be easily verifiable by the verifier. For example, guessing the correct sequence of a combination lock is a proof-of-work, because it is hard to find the correct combination but once produced, can be easily verified; just enter the combination and see if the lock opens.

A proof-of-work is often requested by a server from a client to protect against denial of service attacks by clients. The common scenario is that the client who is requesting a service from a server is challenged by the server to solve a moderately hard puzzle. The client then solves the puzzle and sends the proof-of-work to the server. The server verifies the proof and if accepted, grants the service to the client.

Definition. Let d be a positive number and c, x be bit strings. A function $f(d, c, x) \to \{0, 1\}$ is called a *proof-of-work function* if given d, c, and x, the function can be easily computed, but given d and c, finding x such that $f(d, c, x) = 1$ is computationally difficult but feasible. We refer to d as *difficulty*, to c as *challenge*, and to x as *nonce*. Given d and c, any x such that $f(d, c, x) = 1$ is called a *proof-of-work*.

In bitcoin, for example, the proof-of-work function returns true if and only if SHA-256 cryptographic hash function returns a bit string starting with d zeros in response to the input $x|c$.

21.2 Attacks on Bitcoin

51 Percent Attack. Bitcoin's blockchain can only prevent double-spending of money as long as a majority (i.e., equal or more than 50%) of the computational power in the network is held by honest parties.

Selfish Mining. Eyal and Sirer [ES14] show that the Bitcoin protocol is not incentive-compatible meaning that rational miners prefer to join a pool of *selfish miners*, and the colluding group will increase in size until it becomes a majority. In an attack called the *selfish mining*, the colluding group keeps its discovered blocks private, thereby intentionally forking the chain. When the public branch approaches the group's private branch in length, the selfish miners reveal blocks from their private chain to the public. As a result, honest

[12] Since blocks are mined on average every 10 minutes, 144 blocks are mined per day on average. At 144 blocks per day, 210,000 blocks take on average four years to mine. In the first four years, 10,500,000 bitcoins were mined. Therefore, using a simple geometric series computation, the total circulation will be 21,000,000 bitcoins which will happen after about 95 years from 2009.

miners waste computational resources on mining that end up serving no purpose. While both honest and selfish parties waste some resources, the honest miners waste proportionally more incentivizing rational miners to join the selfish group.

22 Miscellaneous

22.1 Randomized Algorithms

Las Vegas Algorithm. A random algorithm is called *Las Vegas* if it always produces the correct answer. The running time depends on the random choices made in the algorithm. Randomized Quicksort (where pivot elements are chosen at random) is a Las Vegas algorithm because it always sorts the elements correctly, but its running time depends on the choices of pivot.

Monte Carlo Algorithm. A random algorithm is called *Monte Carlo* if it can give the wrong answer sometimes. The running time usually does not depend on the random choices, but for some Monte Carlo algorithms it does.

To turn a Monte-Carlo algorithm into a Las Vegas algorithm, we need to add a means for checking if the answer is correct and retrying if it is not.

22.2 Dining Cryptographers Problem

Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the hotel management for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol: Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see | the one he flipped and the one his left-hand neighbor flipped fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is."

23 Useful Terms and Concepts

Oblivious Transfer (OT). Alice sends one of many pieces of information she has to Bob while she remains oblivious (unaware) as to which piece is transferred. This problem was first proposed by Michael Rabin in 1981 for exchanging secrets between two parties. Later, it found applications in MPC and Private Information Retrieval (PIR).

Non-Interactive Protocol. A protocol is non-interactive if all messages are sent by one party, which is usually called dealer or leader.

Probabilistic-Polynomial Time Algorithm (PPTA). A randomized polynomial-time algorithm. More formally, a polynomial-time algorithm that returns YES with probability more than 1/2 if the answer is YES and returns NO with probability less than or equal to 1/2 if the answer is NO.

Order of an Integer. Let a and n be coprime[13] positive integers. The order of a modulo n denoted by $\text{ord}_n(a)$ is the smallest positive integer k such that

$$a^k \equiv 1 \mod n$$

The existence of such a k is guaranteed by Euler's Theorem.

Euler's Theorem Let a and n be coprime positive integers. We have

$$a^{\varphi(n)} \equiv 1 \mod n$$

where $\varphi(n)$ is number of the positive integers less than or equal to n that are relatively prime to n.

Fast Modular Exponentiation. The fastest algorithm takes about $O(\log^3 n)$ computations.

Side-Channel Attacks. Side-channel attacks allow an adversary to recover information about the input to a cryptographic operation, by measuring something other than the algorithm's result, e.g., power consumption, computation time, or radio-frequency emanations by a device.

Anonymity vs Confidentiality. Anonymity hides *who* performs some action, while confidentiality hides *what* actions are being performed. To guarantee full privacy, an algorithm should often provide both anonymity and confidentiality.

References

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. *A Practical and Provably Secure Coalition-Resistant Group Signature Scheme*, pages 255–270. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[AKS83] M. Ajtai, J. Komlós, and E. Szemerédi. Sorting in $c \log n$ parallel steps. *Combinatorica*, 3(1):1–19, January 1983.

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptol.*, 23(2):281–343, April 2010.

[ALSY06] Man Ho Au, Joseph K. Liu, Willy Susilo, and Tsz Hon Yuen. *Progress in Cryptology - INDOCRYPT 2006: 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006. Proceedings*, chapter Constant-Size ID-Based Linkable and Revocable-iff-Linked Ring Signature, pages 364–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Bat68] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[13] i.e., relatively prime or $\gcd(a, n) = 1$

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 52–61, New York, NY, USA, 1993. ACM.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 1991.

[Ben94] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the Conference on Selected Areas of Cryptography*, pages 120–128, 1994.

[BF01] Dan Boneh and Matt Franklin. *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19–23, 2001 Proceedings*, chapter Identity-Based Encryption from the Weil Pairing, pages 213–229. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[BGT13] Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multiparty computation: how to run sublinear algorithms in a distributed setting. In *Proceedings of the 10th theory of cryptography conference on Theory of Cryptography*, TCC'13, pages 356–376, Berlin, Heidelberg, 2013. Springer-Verlag.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proceedings of the Twentieth ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[Blo16] Blockchain charts: Transactions per block between April 2016 and September 2016, 2016. Available at `https://blockchain.info/charts/n-transactions-per-block`.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 514–532, London, UK, UK, 2001. Springer-Verlag.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM.

[BW86] E Berlekamp and L Welch. Error correction for algebraic block codes, US Patent 4,633,470, December 1986.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, FOCS '01, pages 136–145, Oct 2001.

[CDG88] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 87–119, London, UK, UK, 1988. Springer-Verlag.

[CEG88] David Chaum, Jan-Hendrik Evertse, and Jeroen Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology - EUROCRYPT' 87*, volume 304 of *Lecture Notes in Computer Science*, pages 127–141. Springer Berlin Heidelberg, 1988.

[CFGN96] R. Canetti, U. Friege, O. Goldreich, and M. Naor. Adaptively secure multi-party computation. Technical report, Cambridge, MA, USA, 1996.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, SFCS '85, pages 383–395, Washington, DC, USA, 1985. IEEE Computer Society.

[Cha83] David Chaum. *Blind Signatures for Untraceable Payments*, pages 199–203. Springer US, Boston, MA, 1983.

[Cha85] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[CKLK01] Artur Czumaj, Przemka Kanarek, Krzysztof Lorys, and Miroslaw Kutylowski. Switching networks for generating random permutations, 2001.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 123–132, 2000.

[CR72] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, New York, NY, USA, 1972. ACM.

[CS98] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '98, pages 13–25, London, UK, UK, 1998. Springer-Verlag.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, September 1976.

[DIK$^+$08] I. Damgård, Y. Ishai, M. Krøigaard, J. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. *Advances in Cryptology – CRYPTO '08*, pages 241–261, 2008.

[DKMS14] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking*, volume 8314 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2014.

[DN93] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology — CRYPTO' 92: 12th Annual International Cryptology Conference Santa Barbara, California, USA August 16–20, 1992 Proceedings*, pages 139–147, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: from passive to active security at low cost. In *Proceedings of the 30th annual conference on Advances in cryptology*, CRYPTO'10, pages 558–576, Berlin, Heidelberg, 2010. Springer-Verlag.

[Dou02] John Douceur. The Sybil attack. In *Proceedings of the Second Internation Peer-to-Peer Symposium (IPTPS)*, 2002.

[DPPU86] C. Dwork, D. Peleg, N. Pippenger, and E. Upfal. Fault tolerance in networks of bounded degree. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, page 379. ACM, 1986.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[Elg85] Taher Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, pages 436–454, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, SFCS '87, pages 427–438, Washington, DC, USA, 1987. IEEE Computer Society.

[FLP85] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[FM85] Paul Feldman and Silvio Micali. Byzantine agreement in constant expected time (and trusting no one). In *FOCS*, pages 267–276, 1985.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 148–161, New York, NY, USA, 1988. ACM.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.

[GHY88] Zvi Galil, Stuart Haber, and Moti Yung. Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, pages 135–155, London, UK, UK, 1988. Springer-Verlag.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. *Advances in Cryptology — EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings*, chapter Secure Distributed Key Generation for Discrete-Log Based Cryptosystems, pages 295–310. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Revisiting the distributed key generation for discrete-log based cryptosystems, 2003.

[GKP$^+$13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In *Advances in Cryptology – CRYPTO 2013*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer Berlin Heidelberg, 2013.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *Distributed Computing*, volume 2508 of *Lecture Notes in Computer Science*, pages 17–32. Springer Berlin Heidelberg, 2002.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 365–377, New York, NY, USA, 1982. ACM.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *J. ACM*, 43(3):431–473, May 1996.

[Gol00] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[Gre12] Matthew Green. A few thoughts on cryptographic engineering: Circular security, 2012. URL: http://goo.gl/haBp4q.

[GVZ06] Ronen Gradwohl, Salil P. Vadhan, and David Zuckerman. Random selection with an adversarial majority. In *CRYPTO*, pages 409–426, 2006.

[Hae14] Bernhard Haeupler. Interactive channel capacity revisited. *CoRR*, abs/1408.1467, 2014.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society (New Series)*, 43:439–561, October 2006.

[HR10] Min Huang and Vernon J. Rego. Polynomial evaluation in secret sharing schemes, 2010. URL: http://csdata.cs.purdue.edu/research/PaCS/polyeval.pdf.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272, 2011.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[KR13] Gillat Kol and Ran Raz. Interactive channel capacity. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 715–724, New York, NY, USA, 2013. ACM.

[KS09] Valerie King and Jared Saia. Brief announcement: Fast, scalable Byzantine agreement in the full information model with a nonadaptive adversary. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, PODC '09, pages 304–305, New York, NY, USA, 2009. ACM.

[KS16] Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13:1–13:21, March 2016.

[LP90] Tom Leighton and C. Greg Plaxton. A (fairly) simple circuit that (usually) sorts. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, FOCS '90, pages 264–274, Oct 1990.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer Berlin Heidelberg, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In *Proceedings of the 8th Conference on Theory of Cryptography*, TCC'11, pages 329–346, Berlin, Heidelberg, 2011. Springer-Verlag.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. *Information Security and Privacy: 9th Australasian Conference, ACISP 2004, Sydney, Australia, July 13-15, 2004. Proceedings*, chapter Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups, pages 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Available at https://bitcoin.org/bitcoin.pdf.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 223–238, Berlin, Heidelberg, 1999. Springer-Verlag.

[Ped91a] T. Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology – EUROCRYPT'91*, pages 522–526. Springer, 1991.

[Ped91b] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129–140, London, UK, 1991. Springer-Verlag.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreements in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[RS60] Irving Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, pages 300–304, 1960.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. *Advances in Cryptology — ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings*, chapter How to Leak a Secret, pages 552–565. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 239–252, London, UK, UK, 1990. Springer-Verlag.

[Sch96] Leonard J. Schulman. Coding for interactive communication. *Information Theory, IEEE Transactions on*, 42(6):1745–1756, Nov 1996.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag.

[Ver13] Fré Vercauteren. Final report on main computational assumptions in cryptography. European Network of Excellence in Cryptology (ECRYPT II), 2013.

[Wak68] Abraham Waksman. A permutation network. *J. ACM*, 15(1):159–163, January 1968.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[YL06] Tatu Ylonen and Chris Lonvick. The secure shell (ssh) authentication protocol. RFC 4252, January 2006.

[Zuc97] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Structures and Algorithms*, 11(4):345–367, 1997.