APPENDIX B

Brief Outline of Volume 2

This first volume contains only material on the *basic tools* of modern cryptography, that is, one-way functions, pseudorandomness, and zero-knowledge proofs. These basic tools are used in the construction of the *basic applications* (to be covered in the second volume). The latter will cover encryption, signatures, and general cryptographic protocols. In this appendix we provide brief summaries of the treatments of these basic applications.

B.1. Encryption: Brief Summary

Both private-key and public-key encryption schemes consist of three efficient algorithms: *key generation*, *encryption*, and *decryption*. The difference between the two types of schemes is reflected in the definition of security: The security of a public-key encryption scheme should also hold when the adversary is given the encryption key, whereas that is not required for private-key encryption schemes. Thus, public-key encryption schemes allow each user to broadcast its encryption key, so that any other user can send it encrypted messages (without needing to first agree on a private encryption key with the receiver). Next we present definitions of security for private-key encryption schemes. The public-key analogies can be easily derived by considering adversaries that get the encryption key as additional input. (For private-key encryption schemes, we can assume, without loss of generality, that the encryption key is identical to the decryption key.)

B.1.1. Definitions

For simplicity, we consider only the encryption of a single message; however, this message can be longer than the key (which rules out information-theoretic secrecy [200]). We present two equivalent definitions of security. The first, called *semantic security*, is a computational analogue of Shannon's definition of *perfect secrecy* [200]. The second definition views secure encryption schemes as those for which it is infeasible to distinguish encryptions of any (known) pair of messages (e.g., the all-zeros message

and the all-ones message). The latter definition is technical in nature and is referred to as *indistinguishability of encryptions*.

We stress that the definitions presented here go way beyond saying that it is infeasible to recover the plaintext from the ciphertext. The latter statement is indeed a minimal requirement for a secure encryption scheme, but we claim that it is far too weak a requirement: An encryption scheme typically is used in applications where obtaining specific partial information on the plaintext endangers the security of the application. When designing an application-independent encryption scheme, we do not know which partial information endangers the application and which does not. Furthermore, even if one wants to design an encryption scheme tailored to one's own specific applications, it is rare (to say the least) that one has a precise characterization of all possible partial information that can endanger these applications. Thus, we require that it be *infeasible* to obtain any information about the plaintext from the ciphertext. Furthermore, in most applications the plaintext may not be uniformly distributed, and some a priori information regarding it is available to the adversary. We require that the secrecy of all partial information also be preserved in such a case. That is, even in the presence of a priori information on the plaintext, it is *infeasible* to obtain any (new) information about the plaintext from the ciphertext (beyond what it is feasible to obtain from the a priori information on the plaintext). The definition of semantic security postulates all of this. The equivalent definition of indistinguishability of encryptions is useful in demonstrating the security of candidate constructions, as well as for arguing about their usage as parts of larger protocols.

The Actual Definitions. In both definitions, we consider (feasible) adversaries that obtain, in addition to the ciphertext, auxiliary information that may depend on the potential plaintext (but not on the key). By $E(x)$ we denote the distribution of encryptions of x, when the key is selected at random. To simplify the exposition, let us assume that on security parameter n, the key-generation algorithm produces a key of length n, whereas the scheme is used to encrypt messages of length n^2.

Definition B.1.1 (Semantic Security (Following [123])): *An encryption scheme is* **semantically secure** *if for every feasible algorithm, A, there exists a feasible algorithm B such that for every two functions $f, h : \{0,1\}^* \to \{0,1\}^*$ and all sequences of pairs $(X_n, z_n)_{n\in\mathbb{N}}$, where X_n is a random variable ranging over $\{0,1\}^{n^2}$ and $|z_n|$ is of feasible* (in n) *length,*

$$\Pr[A(E(X_n)h(X_n), z_n) = f(X_n)] < \Pr[B(h(X_n), z_n) = f(X_n)] + \mu(n)$$

where μ is a negligible function. Furthermore, the complexity of B should be related to that of A.

What Definition B.1.1 says is that a feasible adversary does not gain anything by looking at the ciphertext. That is, whatever information (captured by the function f) it tries to compute about the ciphertext when given a priori information (captured by the function h) can essentially be computed as efficiently from the available a priori

information alone. In particular, the ciphertext does not help in (feasibly) computing the least significant bit of the plaintext or any other information regarding the plaintext. This holds for any distribution of plaintexts (captured by the random variable X_n). We now turn to an equivalent definition.

Definition B.1.2 (Indistinguishability of Encryptions (Following [123])): *An encryption scheme has* **indistinguishable encryptions** *if for every feasible algorithm A and all sequences of triples $(x_n, y_n, z_n)_{n\in\mathbb{N}}$, where $|x_n| = |y_n| = n^2$ and $|z_n|$ is of feasible* (in n) *length,*

$$|\mathsf{Pr}[A(E(x_n), z_n) = 1] - \mathsf{Pr}[A(E(y_n), z_n) = 1]| < \mu(n)$$

where μ is a negligible function.

In particular, z_n may equal (x_n, y_n). Thus, it is infeasible to distinguish the encryptions of any two fixed messages such as the all-zeros message and the all-ones message.

Theorem B.1.3: *An encryption scheme is semantically secure if and only if it has indistinguishable encryptions.*

Probabilistic Encryption. It is easy to see that a secure public-key encryption scheme must employ a probabilistic (i.e., randomized) *encryption* algorithm. Otherwise, given the encryption key as (additional) input, it is easy to distinguish the encryption of the all-zeros message from the encryption of the all-ones message. The same holds for private-key encryption schemes when considering the security of encrypting several messages (rather than a single message as done before).[1] This explains the linkage between the foregoing robust security definitions and the *randomization paradigm* (discussed later).

B.1.2. Constructions

Private-key encryption schemes can be constructed based on the existence of one-way functions. In contrast, the known constructions of public-key encryption schemes seem to require stronger assumptions (such as the existence of trapdoor permutations).

B.1.2.1. Private-Key Schemes

It is common practice to use "pseudorandom generators" as a basis for private-key stream ciphers. We stress that this is a very dangerous practice when the "pseudorandom generator" is easy to predict (such as the linear congruential generator or some modifications of it that output a constant fraction of the bits of each resulting number [38, 84]). However, this common practice can become sound provided one uses pseudorandom generators as defined in Section 3.3. Thus, we obtain a *private-key stream cipher* that allows us to encrypt a stream of plaintext bits. Note that such a

[1] Here, for example, using a deterministic encryption algorithm allows the adversary to distinguish two encryptions of the same message from the encryptions of a pair of different messages.

stream cipher does not conform with our formulation of an encryption scheme, since for encrypting several messages we are required to maintain a counter. In other words, we obtain an encryption scheme with a variable state that is modified after the encryption of each message. To obtain a stateless encryption scheme, as in our earlier definitions, we can use a pseudorandom function.

Private-Key Encryption Scheme Based on Pseudorandom Functions. The key-generation algorithm consists of selecting a seed, denoted s, for such a function, denoted f_s. To encrypt a message $x \in \{0,1\}^n$ (using key s), the encryption algorithm uniformly selects a string $r \in \{0,1\}^n$ and produces the ciphertext $(r, x \oplus f_s(r))$. To decrypt the ciphertext (r, y) (using key s), the decryption algorithm just computes $y \oplus f_s(r)$. The proof of security of this encryption scheme consists of two steps (suggested as a general methodology in Section 3.6):

1. Prove that an idealized version of the scheme, in which one uses a uniformly selected function $f : \{0,1\}^n \to \{0,1\}^n$, rather than the pseudorandom function f_s, is secure.
2. Conclude that the real scheme (as presented earlier) is secure (since otherwise one could distinguish a pseudorandom function from a truly random one).

Note that we could have gotten rid of the randomization if we had allowed the encryption algorithm to be history-dependent (e.g., use a counter in the role of r). Furthermore, if the encryption scheme is used for FIFO communication between the parties and both can maintain the counter value, then there is no need for the message sender to transmit the counter value.

B.1.2.2. Public-Key Schemes

Here we use a collection of trapdoor one-way permutations, $\{p_\alpha\}_\alpha$, and a hard-core predicate, b, for it.

The Randomization Paradigm [123]. To demonstrate this paradigm, we first construct a simple public-key encryption scheme.

Key generation: The key-generation algorithm consists of selecting at random a permutation p_α together with a trapdoor for it; the permutation (or rather its description) serves as the public key, whereas the trapdoor serves as the private key.

Encrypting: To encrypt a single bit σ (using public key p_α), the encryption algorithm uniformly selects an element r in the domain of p_α and produces the ciphertext $(p_\alpha(r), \sigma \oplus b(r))$.

Decrypting: To decrypt the ciphertext (y, τ) using the private key, the decryption algorithm simply computes $\tau \oplus b(p_\alpha^{-1}(y))$, where the inverse is computed using the trapdoor (i.e., private key).

This scheme is quite wasteful of bandwidth. However, the paradigm underlying its construction is valuable in practice. For example, it is certainly better to randomly pad messages (say, using padding equal in length to the message) before encrypting them

using RSA than to employ RSA on the plain message. Such a heuristic can be placed on firm ground if the following *conjecture* is supported: Assume that the first $n/2$ least significant bits of the argument constitute a hard-core function of RSA with n-bit-long moduli. Then, encrypting $n/2$-bit messages by padding the message with $n/2$ random bits and applying RSA (with an n-bit modulus) on the result will constitute a secure public-key encryption system, hereafter referred to as *Randomized RSA*.

An alternative public-key encryption scheme is presented in [35]. That encryption scheme augments Construction 3.4.4 (of a pseudorandom generator based on one-way permutations) as follows:

Key generation: As before, the key-generation algorithm consists of selecting at random a permutation p_α together with a trapdoor.

Encrypting: To encrypt the n-bit string x (using public key p_α), the encryption algorithm uniformly selects an element s in the domain of p_α and produces the ciphertext $(p_\alpha^n(s), x \oplus G_\alpha(s))$, where

$$G_\alpha(s) = b(s) \cdot b(p_\alpha(s)) \cdots b\left(p_\alpha^{n-1}(s)\right)$$

(We use the notation $p_\alpha^{i+1}(x) = p_\alpha(p_\alpha^i(x))$ and $p_\alpha^{-(i+1)}(x) = p_\alpha^{-1}(p_\alpha^{-i}(x))$.)

Decrypting: To decrypt the ciphertext (y, z) using the private key, the decryption algorithm first recovers $s = p_\alpha^{-n}(y)$ and then outputs $z \oplus G_\alpha(s)$.

Assuming that factoring Blum integers (i.e., products of two primes each congruent to 3 (mod 4)) is hard, one can use the modular squaring function in the role of the trapdoor permutation and the least significant bit (denoted lsb) in the role of its hard-core predicate [35, 5, 208, 82]. This yields a secure public-key encryption scheme (depicted in Figure B.1) with efficiency comparable to that of RSA. Recall that RSA itself is not secure (as it employs a deterministic encryption algorithm), whereas Randomized

Private key: Two $n/2$-bit-long primes, p and q, each congruent to 3 (mod 4).
Public key: Their product $N \stackrel{\text{def}}{=} pq$.
Encryption of message $x \in \{0,1\}^n$:

1. Uniformly select $s_0 \in \{1, \ldots, N\}$.
2. For $i = 1, \ldots, n+1$, compute $s_i \leftarrow s_{i-1}^2 \bmod N$ and $\sigma_i = \text{lsb}(s_i)$.

The ciphertext is (s_{n+1}, y), where $y = x \oplus \sigma_1\sigma_2 \cdots \sigma_n$.
Decryption of the ciphertext (r, y):
Precomputed: $d_p = ((p+1)/4)^n \bmod p-1$, $d_q = ((q+1)/4)^n \bmod q-1$, $c_p = q \cdot (q^{-1} \bmod p)$, and $c_q = p \cdot (p^{-1} \bmod q)$.

1. Let $s' \leftarrow r^{d_p} \bmod p$ and $s'' \leftarrow r^{d_q} \bmod q$.
2. Let $s_1 \leftarrow c_p \cdot s' + c_q \cdot s'' \bmod N$.
3. For $i = 1, \ldots, n$, compute $\sigma_i = \text{lsb}(s_i)$ and $s_{i+1} \leftarrow s_i^2 \bmod N$.

The plaintext is $y \oplus \sigma_1\sigma_2 \cdots \sigma_n$.

Figure B.1: The Blum-Goldwasser public-key encryption scheme [35].

RSA (defined earlier) is not known to be secure under standard assumptions such as the intractability of factoring (or of inverting the RSA function).[2]

B.1.3. Beyond Eavesdropping Security

The foregoing definitions refer only to a "passive" attack in which the adversary merely eavesdrops on the communication line (over which ciphertexts are being sent). Stronger types of attacks, culminating in the so-called chosen ciphertext attack, may be possible in various applications. Furthermore, these definitions refer to an adversary that tries to extract explicit information about the plaintext. A less explicit attempt, captured by the so-called notion of *malleability*, is to generate an encryption of a related plaintext (possibly without learning anything about the original plaintext). Thus, we have a "matrix" of adversaries, with one dimension (parameter) being the *type of attack* and the second being its *purpose*.

Types of Attacks. The following mini-taxonomy of attacks certainly is not exhaustive:

1. *Passive attacks*, as captured in the foregoing definitions. Among public-key schemes, we distinguish two sub-cases:
   - **(a)** A *key-oblivious* passive attack, as captured in the foregoing definitions. By "key-obliviousness" we refer to the fact that the choice of plaintext does not depend on the public key.
   - **(b)** A *key-dependent* passive attack, in which the choice of plaintext may depend on the public key.

   (In Definition B.1.1 the choice of plaintext means the random variable X_n, whereas in Definition B.1.2 it means the pair of strings (x_n, y_n). In both of these definitions, the choice of the plaintext is non-adaptive.)
2. *Chosen plaintext attacks*. Here the attacker can obtain the encryption of any plaintext of its choice (under the key being attacked). Such an attack does not add power in case of public-key schemes.
3. *Chosen ciphertext attacks*. Here the attacker can obtain the decryption of any ciphertext of its choice (under the key being attacked). That is, the attacker is given oracle access to the decryption function corresponding to the decryption key in use. We distinguish two types of such attacks:
   - **(a)** In an *a-priori-chosen* ciphertext attack, the attacker is given this oracle access prior to being presented the ciphertext that it will attack (i.e., the ciphertext for which it has to learn partial information or form a related ciphertext). That is, the attack consists of two stages: In the first stage the attacker is given the oracle access, and in the second stage the oracle is removed and the attacker is given a "test ciphertext" (i.e., a target to be learned or modified in violation of non-malleability).

[2]Recall that Randomized RSA is secure assuming that the $n/2$ least significant bits constitute a hard-core function for n-bit RSA moduli. We only know that the $O(\log n)$ least significant bits constitute a hard-core function for n-bit moduli [5].

(b) In an *a-posteriori-chosen* ciphertext attack the attacker is given the target ciphertext first, but its access to the oracle is restricted in that it is not allowed to make a query equal to the target ciphertext.

In both cases, the adversary can make queries that do not correspond to a legitimate ciphertext, and the answers will be accordingly (i.e., a special "failure" symbol).

Purpose of Attacks. Again, the following is not claimed to be exhaustive:

1. Standard *security*: the infeasibility of *obtaining information regarding the plaintext*. As defined earlier, such information must be a function (or a randomized process) applied to the bare plaintext and cannot depend on the encryption (or decryption) key.
2. In contrast, the notion of *non-malleability* [64] refers to generating a string depending on both the plaintext and the current encryption key. Specifically, one requires that it be infeasible for an adversary, given a ciphertext, to produce a valid ciphertext for a related plaintext. For example, given a ciphertext of a plaintext of the form $1x$, it should be infeasible to produce a ciphertext to the plaintext $0x$.

With the exception of passive attacks on private-key schemes, non-malleability always implies security against attempts to obtain information on the plaintext. Security and non-malleability are equivalent under a-posteriori-chosen ciphertext attack (cf. [64, 16]). For a detailed discussion of the relationships among the various notions of secure private-key and public-key encryptions, the reader is referred to [142] and [16], respectively.

Some Known Constructions. As in the basic case, the (strongly secure) private-key encryption schemes can be constructed based on the existence of one-way functions, whereas the (strongly secure) public-key encryption schemes are based on the existence of trapdoor permutations.

Private-key schemes: The private-key encryption scheme based on pseudorandom functions (described earlier) is secure also against a-priori-chosen ciphertext attacks.[3]

It is easy to turn any passively secure private-key encryption scheme into a scheme secure under (a posteriori) chosen ciphertext attacks by using a message-authentication scheme[4] on top of the basic encryption.

Public-key schemes: Public-key encryption schemes secure against a-priori-chosen ciphertext attacks can be constructed assuming the existence of trapdoor permutations and utilizing non-interactive zero-knowledge proofs see [176]. (Recall that the latter proof systems can be constructed under the former assumption.)

[3]Note that this scheme is not secure under an a-posteriori-chosen ciphertext attack: On input a ciphertext $(r, x \oplus f_s(r))$, we obtain $f_s(r)$ by making the query (r, y'), where $y' \neq x \oplus f_s(r)$. (This query is answered with x' such that $y' = x' \oplus f_s(r)$.)

[4]See definition in Section B.2.

Public-key encryption schemes secure against a-posteriori-chosen ciphertext attacks can also be constructed under the same assumption [64], but this construction is even more complex.

In fact, both constructions of *public-key* encryption schemes secure against chosen ciphertext attacks are to be considered as plausibility results (which also offer some useful construction paradigms). Presenting "reasonably efficient" public-key encryption schemes that are secure against (a posteriori) chosen ciphertext attacks, under widely believed assumptions, is an important open problem.[5]

B.1.4. Some Suggestions

B.1.4.1. Suggestions for Further Reading

Fragments of a preliminary draft for the intended chapter on encryption schemes can be obtained online [99].

In addition, there are the original papers: There is a good motivating discussion in [123], but we prefer the definitional treatment of [92, 94], which can be substantially simplified if one adopts non-uniform complexity measures (as done above).[6] Further details on the construction of public-key encryption schemes (sketched above) can be found in [123, 92, 35, 5]. For discussion of non-malleable cryptography, which actually transcends the domain of encryption, see [64].

B.1.4.2. Suggestions for Teaching

We suggest a focus on the basic notion of security (treated in Sections B.1.1 and B.1.2): Present both definitions, prove their equivalence, and discuss the need to use *randomness* during the encryption process in order to meet these definitions. Next, present all constructions described in Section B.1.2. We believe that the draft available online [99] provides sufficient details for all of these.

B.2. Signatures: Brief Summary

Again, there are private-key and public-key versions, both consisting of three efficient algorithms: *key generation*, *signing*, and *verification*. (Private-key signature schemes are commonly referred to as *message-authentication schemes* or *codes* (MAC).) The difference between the two types is again reflected in the definitions of security. This difference yields different functionalities (even more than in the case of encryption): Public-key signature schemes (hereafter referred to as signature schemes) can be used to produce signatures that are *universally verifiable* (given access to the public key of the signer). Private-key signature schemes (hereafter referred to as message-authentication schemes) typically are used to authenticate messages sent

[5] The "reasonably efficient" scheme of [57] is based on a strong assumption regarding the Diffie-Hellman key exchange. Specifically, it is assumed that for a prime P and primitive element g, given $(P, g, (g^x \bmod P), (g^y \bmod P), (g^z \bmod P))$, it is infeasible to decide whether or not $z \equiv xy \pmod{P-1}$.

[6] We comment that [92] follows [94] in providing a uniform-complexity treatment of the security of encryption schemes.

among a (small) set of *mutually trusting* parties (since the ability to verify signatures may be linked to the ability to produce them). In other words, message-authentication schemes are used to authenticate information sent between (typically two) parties, and the purpose is to *convince the receiver* that the information has indeed been sent by the legitimate sender. In particular, message-authentication schemes cannot *convince a third party* that the sender has indeed sent the information (rather than the receiver having generated it by itself). In contrast, public-key signatures can be used to convince third parties: A signature to a document typically is sent to a second party, so that in the future that party can (by merely presenting the signed document) convince third parties that the document was indeed generated/sent/approved by the signer.

B.2.1. Definitions

We consider very powerful attacks on the signature scheme as well as a very liberal notion of breaking it. Specifically, the attacker is allowed to obtain signatures to any message of its choice. One may argue that in many applications such a general attack is not possible (since messages to be signed must have a specific format). Yet our view is that it is impossible to define a general (i.e., application-independent) notion of admissible messages, and thus it seems that a general/robust definition of an attack must be formulated as suggested here. (Note that, at worst, our approach is overly cautious.) Likewise, the adversary is said to be successful if it can produce a valid signature to ANY message for which it has not asked for a signature during its attack. Again, this defines the ability to form signatures to possibly "non-sensical" messages as a breaking of the scheme. Yet, again, we see no way to have a general (i.e., application-independent) notion of "meaningful" messages (so that only forging signatures to them would be consider a breaking of the scheme).

Definition B.2.1 (Unforgeable Signatures [125]):

- *A* **chosen message attack** *is a process that on input a verification key can obtain signatures* (relative to the corresponding signing key) *to messages of its choice.*
- *Such an attack is said to* **succeed** (in existential forgery) *if it outputs a valid signature to a message for which it has* not *requested a signature during the attack.*
- *A signature scheme is* **secure** (or unforgeable) *if every* feasible *chosen message attack succeeds with at most negligible probability.*

We stress that *plain* RSA (like plain versions of Rabin's scheme [187] and DSS [169]) is not secure under the foregoing definition. However, it may be secure if the message is "randomized" before RSA (or another scheme) is applied [22]. Thus the randomization paradigm seems pivotal here too.

The definition of security for message-authentication schemes is similar, except that the attacker does not get the verification key as input.

B.2.2. Constructions

Both message-authentication and signature schemes can be constructed based on the existence of one-way functions.

B.2.2.1. Message Authentication

Message-authentication schemes can be constructed using pseudorandom functions [103]: To authenticate the message x with respect to key s, one generates the tag $f_s(x)$, where f_s is the pseudorandom function associated with s. Verification is done in the same (analogous) way. However, as noted in [15], *extensive* use of pseudorandom functions would seem to be overkill for achieving message authentication, and more efficient schemes can be obtained based on other cryptographic primitives. We mention two approaches:

1. *fingerprinting* the message using a scheme that is *secure against forgery provided that the adversary does not have access to the scheme's outcome* (e.g., using Universal Hashing [49]), and "*hiding*" the result using a *non-malleable* scheme (e.g., a private-key encryption or a pseudorandom function). (Non-malleability is not required in certain cases [209].)
2. *hashing* the message *using a collision-free scheme* [58, 59] and *authenticating* the result using a MAC that operates on (short) fixed-length strings [15].

B.2.2.2. Signature Schemes

Three central paradigms in the construction of signature schemes are the "refreshing" of the "effective" signing key, the use of an "authentication tree," and the "hashing paradigm."

The Refreshing Paradigm [125]. To demonstrate this paradigm, suppose we have a signature scheme that is robust against a "random message attack" (i.e., an attack in which the adversary obtains signatures only to randomly chosen messages). Further suppose that we have a *one-time* signature scheme (i.e., a signature scheme that is secure against an attack in which the adversary obtains a signature to a *single* message of its choice). Then we can obtain a secure signature scheme as follows: When a new message needs to be signed, we generate a new random signing key for the one-time signature scheme, use it to sign the message, and sign the corresponding (one-time) verification key using the fixed signing key of the main signature scheme[7] (which is robust against a "random message attack") [71]. We note that one-time signature schemes (as utilized here) are easy to construct (e.g., [161]).

The Authentication-Tree Paradigm [160, 125]. To demonstrate this paradigm, we show how to construct a general signature scheme using only a one-time signature scheme (alas, one where a $2n$-bit string can be signed with respect to an n-bit-long

[7]Alternatively, one can generate the one-time key pair and the signature to its verification key ahead of time, leading to an "off-line/on-line" signature scheme [71].

verification key). The idea is to use the initial signing key (i.e., the one corresponding to the public verification key) in order to sign/authenticate two new/random verification keys. The two corresponding signing keys are used to sign/authenticate four new/random verification keys (two per each signing key), and so on. Stopping after ℓ such steps, this process forms a binary tree with 2^ℓ leaves, where each leaf corresponds to an instance of the one-time signature scheme. The signing keys at the leaves can be used to sign the actual messages, and the corresponding verification keys can be authenticated using the path from the root. That is, to sign a new message, we proceed as follows:

1. Allocate a new leaf in the tree. This requires either keeping a counter of the number of messages signed thus far or selecting a leaf at random (assuming that the number of leaves is much larger than the square of the number of messages to be signed).
2. Generate or retrieve from storage the pairs of signing/verification keys corresponding to each vertex on the path from the root to the selected leaf, along with the key pairs of the siblings of the vertices on the path. That is, let $v_0, v_1, \ldots, v_\ell$ denote the vertices along the path from the root v_0 to the selected leaf v_ℓ, and let u_i be the sibling of v_i (for $i = 1, \ldots, \ell$). Then we generate/retrieve the key pairs of each v_i and each u_i, for $i = 1, \ldots, \ell$.

   It is important to use the same key pair when encountering the same vertex in the process of signing two different messages.
3. Sign the message using the signing key associated with the selected leaf v_ℓ. Sign each pair of verification keys associated with the children of each internal vertex, along the foregoing path, using the signing key associated with the parent vertex. That is, for $i = 1, \ldots, \ell$, sign the verification keys of v_i and u_i (placed in some canonical order) using the signing key associated with vertex v_{i-1}.

   The signature is obtained by concatenating all these signatures (along with the corresponding verification keys). Recall that the key pair associated with the root is the actual key pair of the signature scheme; that is, the verification component is placed in the public file, and the signature of the verification keys of the root's children (relative to the root's signing key) is part of all signatures.

Pseudorandom functions can be used to eliminate the need to store the values of vertices used in previous signatures [89].

Employing this paradigm, and assuming that the RSA function is infeasible to invert, one obtains a secure signature scheme [125, 89] (with a counter of the number of messages signed) in which the ith message is signed/verified in time $2 \log_2 i$ slower than plain RSA. Using a tree of large fan-in (and assuming again that RSA is infeasible to invert), one can obtain a secure signature scheme [67, 56] that for reasonable parameters is only five times slower than plain RSA.[8] We stress that plain RSA is not a secure signature scheme, whereas the security of its randomized version (mentioned earlier) is not known to be reducible to the assumption that RSA is hard to invert.

[8] This figure refers to signing up to 1,000,000,000 messages. The scheme in [67] requires a universal set of system parameters consisting of 1000–2000 integers of the size of the moduli. In the scheme of [56], that requirement is removed.

The Hashing Paradigm. A common practice is to sign real documents via a two-stage process: First the document is hashed into a (relatively) short bit string, and then the basic signature scheme is applied to the resulting string. We note that this heuristic becomes sound provided the hashing function is *collision-free* (as defined in [58]). Collision-free hashing functions can be constructed, assuming the existence of claw-free collections (as in Definition 2.4.6) [58]. One can indeed postulate that certain off-the-shelf products (e.g., MD5 or SHA) are collision-free, but such assumptions need to be tested (and indeed may turn out false). We stress that using a hashing scheme in the foregoing two-stage process without carefully evaluating whether or not it is collision-free is a very dangerous practice.

One useful variant on the foregoing paradigm is the use of *universal one-way hashing functions* (as defined in [175]), rather than the collision-free hashing used earlier. In such a case, a new hashing function is selected for each application of the scheme, and the basic signature scheme is applied both to the (succinct) description of the hashing function and to the resulting (hashed) string. (In contrast, when using a collision-free hashing function, the same function, the description of which is part of the signer's public key, is used in all applications of the signature scheme.) The advantage of using universal one-way hashing functions is that their security requirement seems weaker than that for the collision-free condition (e.g., the former can be constructed using any one-way function [192], whereas this is NOT known for the latter).

Theorem B.2.2 (Plausibility Result [175, 192]): *Signature schemes exist if and only if one-way functions exist.*

Unlike the paradigms (and some of the constructions) described earlier, the known construction of signature schemes from *arbitrary* one-way functions has no practical significance. It is indeed an important open problem to provide an alternative construction that can be practical and still utilize an *arbitrary* one-way function.

B.2.3. Some Suggestions

B.2.3.1. Suggestions for Further Reading

Fragments of a preliminary draft for the intended chapter on signature schemes can be obtained on line [100].

In addition, there are the original papers: For a definitional treatment of *signature schemes*, the reader is referred to [125] and [183]. Easy-to-understand constructions appear in [20, 71, 67]. The proof of Theorem B.2.2 can be extracted from [175, 192]: The first paper presents the basic approach and implements it using any one-way permutation, whereas the second paper shows how to implement this approach using any one-way function. Variants on the basic model are discussed in [183] and in [50, 137]. For discussion of *message-authentication schemes* (MACs) the reader is referred to [15].

B.2.3.2. Suggestions for Teaching

We suggest a focus on signature schemes, presenting the main definition and some construction. One may use [125] for the definitional treatment, but should *not* use

it for the construction, the underlying ideas of which are more transparent in papers such as [20] and [175]. Actually, we suggest presenting a variant on the signature scheme of [175], using collision-free hashing (cf. [58]) instead of universal one-way hashing (cf. [175]). This allows one to present, within a few lectures, many important paradigms and techniques (e.g., the refreshing paradigm, authentication trees, the hashing paradigm, and one-time signature schemes). We believe that the draft available online [100] provides sufficient details for such a presentation.

A basic treatment of message authentication (i.e., motivation, definition, and construction based on pseudorandom functions) can be presented within one lecture, and [100] can be used for this purpose too. (This, however, will not cover alternative approaches employed toward the construction of more efficient message-authentication schemes.)

B.3. Cryptographic Protocols: Brief Summary

A general framework for casting cryptographic (protocol) problems consists of specifying a random process that maps n inputs to n outputs. The inputs to the process are to be thought of as local inputs of n parties, and the n outputs are their corresponding local outputs. The random process describes the desired functionality. That is, if the n parties were to trust each other (or trust some outside party), then each could send its local input to the trusted party, who would compute the outcome of the process and send each party the corresponding output. The question addressed in this section is the extent to which this trusted party can be "emulated" by the mutually distrustful parties themselves.

B.3.1. Definitions

For simplicity, we consider the special case where the specified process is deterministic and the n outputs are identical. That is, we consider an arbitrary n-ary function and n parties that wish to obtain the value of the function on their n corresponding inputs. Each party wishes to obtain the correct value of the function and prevent any other party from gaining anything else (i.e., anything beyond the value of the function and what is implied by it).

We first observe that (one thing that is unavoidable is that) each party can change its local input before entering the protocol. However, this is also unavoidable when the parties utilize a trusted party. In general, the basic paradigm underlying the definitions of *secure multi-party computations*[9] amounts to saying that situations that may occur in the real protocol can be simulated in an ideal model (where the parties can employ a trusted party). Thus, the "effective malfunctioning" of parties in secure protocols is restricted to what is postulated in the corresponding ideal model. The specific definitions

[9] Our current understanding of the definitional issues is most indebted to the high-level discussions in the unfinished manuscript of [165]. A similar definitional approach is presented in [11, 12]. The approach of [122] is more general: It avoids the definition of security (w.r.t a given functionality) and defines instead a related notion of protocol robustness. One minimalistic instantiation of the definitional approach of [165, 11, 12] is presented in [45] and is shown to satisfy the main conceptual concerns.

differ in the specific restrictions and/or requirements placed on the parties in the real computation. This typically is reflected in the definition of the corresponding ideal model; see the examples that follow.

B.3.1.1. An Example: Computations with Honest Majority

Here we consider an ideal model in which any minority group (of the parties) can collude as follows. First, this minority shares its original inputs and decides together on replacement inputs[10] to be sent to the trusted party. (The other parties send their respective original inputs to the trusted party.) When the trusted party returns the output, each majority player outputs it locally, whereas the colluding minority can compute an output based on all they know (i.e., the output and all the local inputs of these parties). A *secure multi-party computation with honest majority* is required to simulate this ideal model. That is, the effect of any feasible adversary that controls a minority of the players in the actual protocol can essentially be simulated by a (different) feasible adversary that controls the corresponding players in the ideal model. This means that in a secure protocol the effect of each minority group is "essentially restricted" to replacing its own local inputs (independently of the local inputs of the majority players) before the protocol starts and replacing its own local outputs (depending only on its local inputs and outputs) after the protocol terminates. (We stress that in the real execution the minority players do obtain additional pieces of information; yet in a secure protocol they gain essentially nothing from these additional pieces of information.)

Secure protocols according to this definition can even tolerate a situation where a minority of the parties choose to abort the execution. An aborted party (in the real protocol) is simulated by a party (in the ideal model) that aborts the execution either before supplying its input to the trusted party (in which case a default input is used) or after supplying its input. In either case, the majority players (in the real protocol) are able to compute the output even though a minority aborted the execution. This cannot be expected to happen when there is no honest majority (e.g., in a two-party computation) [53].

B.3.1.2. Another Example: Two-Party Computations Allowing Abort

In light of the foregoing, we consider an ideal model where each of the two parties can "shut down" the trusted (third) party at any point in time. In particular, this can happen after the trusted party has supplied the outcome of the computation to one party but before it has supplied it to the second. A *secure two-party computation allowing abort* is required to simulate this ideal model. That is, each party's "effective malfunctioning" in such a secure protocol is restricted to supplying an initial input of its choice and aborting the computation at any point in time. We stress that, as before, the choice of the initial input of each party cannot depend on the input of the other party.

[10] Such replacement can be avoided if the local inputs of parties are verifiable by the other parties. In such a case, a party (in the ideal model) has the choice of either joining the execution of the protocol with its correct local input or not joining the execution at all (but it cannot join with a replaced local input). Secure protocols simulating this ideal model can be constructed as well.

Generalizing the preceding, we can consider *secure multi-party computation allowing abort*. Here, in the ideal model, each of the parties can "shut down" the trusted party at any point in time; in particular, this can happen after the trusted party has supplied the outcome of the computation to some but not all of the parties.

B.3.2. Constructions

Theorem B.3.1 (General Plausibility Results, Loosely Stated): *Suppose that trapdoor permutations exist. Then*

- *any multi-party functionality can be securely computed in a model allowing abort* (cf. [211] for the two-party case and [113] for the case of more than two parties).
- *any multi-party functionality can be securely computed provided that a strict majority of the parties are honest* [112, 113].

The proof of each item proceeds in two steps [98]:

1. Presenting secure protocols for a "semi-honest" model in which the bad parties follow the protocol, except that they also keep a record of all intermediate results.[11]

One key idea is to consider the propagation of values along the wires of a circuit (which computes the desired function), going from the input wires to the output wires. The execution of these protocols starts by each party sharing its inputs with all other parties, using a secret sharing scheme, so that any strict subset of the shares yields no information about the secret (e.g., each party is given a uniformly chosen share, and the dealer's share is set to the XOR of all other shares). A typical step consists of the secure computation of shares of the output wire of a gate from the shares of the input wires of this gate. That is, the m parties employ a secure protocol for computing the randomized m-party functionality $((a_1, b_1), \ldots, (a_m, b_m)) \mapsto (c_1, \ldots, c_m)$, where the c_i's are uniformly distributed subject to $\oplus_{i=1}^m c_i = \text{gate}(\oplus_{i=1}^m a_i, \oplus_{i=1}^m b_i)$. Repeating this step for each gate of the circuit (in a suitable order), the parties securely propagate shares along the wires of the circuit, going from the input wires of the circuit to its output wires. At the end of this propagation process, each party announces its shares in the output wires of the circuit, and the actual output is formed. Thus, securely computing an arbitrary functionality (which may be quite complex) is reduced to securely computing a few specific simple functionalities (i.e., given shares for the inputs of a Boolean gate, securely compute random shares for the output of this gate). Indeed, secure protocols for computing these simple functionalities are also provided.

2. Transforming protocols secure in the "semi-honest" model into full-fledged secure protocols. Here zero-knowledge proofs and protocols for fair coin-tossing are used in order to "force" parties to behave properly (i.e., as in the "semi-honest" model).

Fair coin-tossing protocols are constructed using non-oblivious commitment schemes (see Section 4.9.2), which in turn rely on zero-knowledge proofs of knowledge (see Section 4.7).

[11] In other words, we need to simulate the local views of the dishonest players when given only the local inputs and outputs of the honest players. Indeed, this model corresponds to the honest-verifier model of zero-knowledge (see Section 4.3.1.7).

We stress the general nature of these constructions and view them as plausibility results asserting that a host of cryptographic problems are solvable, assuming the existence of trapdoor permutations. As discussed in the case of zero-knowledge proofs, the value of these general results is in allowing one to easily infer that the problem he/she faces is solvable in principle (as typically it is easy to cast problems within this framework). However, we do not recommend using (in practice) the solutions derived by these general results; one should rather focus on the specifics of the problem at hand and solve it using techniques and/or insights available from these general results.[12]

Analogous plausibility results have been obtained in a variety of models. In particular, we mention secure computations in the private-channels model [27, 51] and in the presence of mobile adversaries [182].

B.3.3. Some Suggestions

B.3.3.1. Suggestions for Further Reading

A draft of a manuscript that is intended to cover this surveyed material is available online from [98]. The draft provides an exposition of the basic definitions and results, as well as detailed proofs for the latter. More refined discussions of definitional issues can be found in [11, 12, 44, 45, 122, 165]; our advice is to start with [45].

B.3.3.2. Suggestions for Teaching

This area is very complex, and so we suggest that one merely present *sketches* of some definitions and constructions. Specifically, we suggest picking one of the two settings (i.e., computation with honest majority or two-party computation) and sketching the definition and the construction. Our own choice would be the two-party case; alas, the definition (allowing abort) is more complicated (but this is more than compensated for by simpler notation and a simpler construction that relies on relatively fewer ideas). We suggest emphasizing the definitional approach (i.e., "emulating a trusted party" as simulation of any adversary operating in the real model by an ideal-model adversary) and presenting the main ideas underlying the construction (while possibly skipping a few). We believe that the draft available online from [98] provides sufficient details for all of these.

[12] For example, although Threshold Cryptography (cf., [62, 87]) is merely a special case of multi-party computation, it is indeed beneficial to focus on its specifics.