On the Use of Complexity in Cryptography

Oded Goldreich[*]

A computational complexity gap, captured in the definition of one-way functions, is a necessary and sufficient condition for much of modern cryptography. Loosely speaking, one-way functions are functions that are easy to compute but hard to invert (in an average-case sense). The existence of one-way functions implies that P is different than NP, which means that such a complexity gap is only wide conjectured to exist (rather than known for a fact). We demonstrate the use of this gap in the case of two basic cryptographic tasks: The tasks of providing secret and authenticated communication, which in turn are reduced to the construction of encryption and signature schemes.

Encryption schemes. Such schemes are supposed to provide secret communication between parties in a setting in which these parties communicate over a channel that may be eavesdropped by an adversary. There are two cases differing according to whether or not the communicating parties have agreed on a common secret prior to the communication. In both cases, the encryption scheme consists of three efficient procedures: *key generation*, *encryption* (denoted by E), and *decryption* (D). Loosely speaking, on input a security parameter n, the key-generation procedure outputs a (random) pair of corresponding (n-bit long) encryption and decryption keys, (e,d), such that for every bit string x, it holds that $D_d(E_e(x)) = x$, where $E_e(x)$ (resp., $D_d(y)$) denotes the output of the encryption (resp., decryption) procedure on input (e,x) (resp., (d,y)).

The difference between the two cases lies in the way in which the scheme is employed and this will be reflected in the definition of security. In the first case, known as the *private-key* case, a set of mutually trustful parties jointly employ the key-generation process, prior to the actual communication, obtaining a pair of keys (e,d). We stress that, in this case, the encryption key e is known to all trusted parties and to them only. Later, each trusted party may encrypt messages by applying E_e, and retrieve them (i.e., decrypt) by applying D_d. The information available to the adversary, in this case, is a sequence of encrypted messages, sent over the channel, using a fixed encryption key unknown to it. (We stress that the total amount of information encrypted using this encryption key may be much greater than the length of the key, and so perfect information theoretic secrecy is not possible).

In the second case, known as the *public-key* case, the receiver invokes the key-generation process, publicizes the encryption key e (but not the decryption key d), and the sender uses e to generate encryptions as before. This allows everybody (not only parties that the receiver trusts) to send encrypted messages to the receiver; however, in such a case the adversary also knows the encryption key e. Thus, the information available to the adversary in this case is a sequence of encrypted messages, sent over the channel, using a fixed encryption key that is also known to it. In both cases, security amounts to asserting that it is infeasible for the adversary to learn anything from the information available to it. That is, whatever the adversary can efficiently compute from the

[*]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. `oded.goldreich@weizmann.ac.il`

public information, can be efficiently computed from scratch.[1]

Note that in the private-key case, we may assume, without loss of generality, that $e = d$; whereas in the public-key case, d must be hard to compute from e. Private-key encryption schemes exist if and only if one-way functions exists. Public-key encryption schemes can be constructed based on a seemingly stronger assumption; yet this assumption is implied by widely believed conjectures such as the conjectured intractability of factoring integers.

Signature schemes. Here too we have two cases corresponding to whether or not a certain key (here it is the verification key) is public. In both cases, the scheme consists of three probabilistic polynomial-time procedures: *key generation*, *signing* (S), and *verification* (V). On input a security parameter n, the key-generation procedure outputs a (random) pair of corresponding (n-bit long) signing and verification keys, (s, v), such that for every bit string x, it holds that $V_v(x, S_s(x)) = 1$, where $S_s(x)$ (resp., $V_d(x, y)$) denotes the output of the signing (resp., verification) procedure on input (s, x) (resp., (v, x, y)).

The difference between the two cases lies in the way in which the scheme is employed and this will be reflected in the definition of security. In the *private-key* case (also known as message-authentication), the scheme is used to authenticate messages sent between mutually trustful parties that communicate over a channel that may be subject to message corruptions (and/or message insertion/deletion). It is assumed that the parties have jointly invoked the key-generation process prior to the communication, obtaining a signing key s (which may, w.l.o.g, equal the verification key v). Subsequently, the sender authenticates each message x by appending $S_s(x)$ to it, and the receiver verifies the authenticity by applying V_v. In the *public-key* case, the scheme is used in order to allow universal verification of commitments made by parties. To this end, each party invokes the key-generation process, deposits the resulting verification key v on a trusted public-file, and keeps the corresponding signing key s secret. When the user later wishes to commit to a document, it applies S_s to it, and this commitment is universally verifiable with respect to its public verification-key.

In both cases, security amounts to asserting that it is infeasible for anybody given the public information (but not having the signing key), to produce a valid signature (i.e., a commitment with respect to the verification key) to a document for which such a commitment was not previously supplied by a party holding the signing-key. That is, forgery should be infeasible even if the forger may ask the legitimate user to sign documents of its choice; after such an attack the forger may indeed present valid signatures to all documents for which it has requested a signature, but not to any other document. (We stress that in case of public-key schemes this is required to hold even if the forger has the verification key).

As in the case of encryption, in the private-key case, we may assume, without loss of generality, that $v = s$; whereas in the public-key case, s must be hard to compute from v. Both private-key and public-key signature schemes exist if and only if one-way functions exists.

Beyond encryption and signature schemes. We stress that cryptography encompasses much more than methods for providing secret and authenticated communication. In general, cryptography is concerned with the construction of schemes that maintain *any* desired functionality under malicious attempts aimed at making these schemes deviate from their prescribed functionality.

[1]The actual formulation refers to the notion of computational indistinguishability. It asserts that for every distribution ensemble of the first type (representing that which the adversary computes from the information available to it) there exists a distribution ensemble of the second type (representing that which can be computed from scratch) so that the two ensembles are computationally indistinguishable.

Loosely speaking, a secure implementation of a multi-party functionality is a multi-party protocol in which the *impact* of malicious parties is effectively restricted to application of the prescribed functionality to inputs chosen by the corresponding parties. One major result in this area states that, under plausible assumptions regarding computational difficulty, *any efficiently computed functionality can be securely implemented.*