1 Symmetric Cryptography

1.1 Stream Ciphers and Block Ciphers

Cryptographers have long been designing things called "stream ciphers" and "block ciphers." A stream cipher (e.g., RC4 [Riv87]) takes an input key (also known as seed) and produces a long (usually unlimited) stream of random-looking bits. A block cipher (e.g., DES [NIS77]) takes a key and an input, and produces an output of the same length as the input. For each key, a block cipher is a permutation.

While stream ciphers and block ciphers predate modern cryptographic notions, today people most often model them as pseudorandom generations and pseudorandom permutations, respectively. Note that stream ciphers and block ciphers used in practice are not provably pseudorandom generations and functions; rather, they are *believed* to have these properties (although some think that this is too strong of an assumption).

We already designed pseudorandom generators that are provably secure under a reasonable assumption. We will do the same for pseudorandom functions and permutations below (after defining them, of course). It is therefore legitimate to ask why people use unprovable designs when so many provably secure ones are available. The answer is mainly speed. As we will see shortly, traditional stream and block ciphers are orders of magnitude faster than provable ones. Thus, they are preferred for encrypting bulk data, particularly by computationally weak devices in real time (cell phones, wireless network cards, etc.).

We will spend this and next lecture understanding what pseudorandom generators and functions are and how to build them (provably!). Note that our provably secure constructions will be of interest mainly as "feasibility" results: we show that it can be done, but most people will not use our provable constructions. Instead, in practice they will simply opt for assuming that RC4 is a pseudorandom generator and DES is a pseudorandom function, even though these assumptions are seemingly stronger than simply assuming that factoring is hard.

Nonetheless, whether you use a provable pseudorandom generator, such as Blum-Blum-Shub, or a heuristic one, such as RC4, you still need to use it right. Therefore, we will then turn to understanding how to *use* pseudorandom generators and functions to accomplish actual goals (privacy and authenticity).

But first, by way of example, we briefly study RC4 and DES.

1.2 RC4

RC4, designed by Rivest, is a stream cipher that takes a key and produces a long stream of random-looking bits. The key is used to initialize an array S of 256 elements that contains each byte from 0 to 255 exactly once. The key is also used to initialize two indices, i and j, into the array. We do not describe the initialization step here. After the initialization, the following steps take place. All operations below are modulo 256

Repeat for as many times as the number of output bytes needed:

1. $i \leftarrow i + 1$
2. $j \leftarrow j + S[i]$
3. Swap $S[i]$ and $S[j]$
4. Output $S[S[i] + S[j]]$

Thus, this stream cipher outputs 8 bits per 3 byte additions and 3 array lookups. It requires a tiny amount of memory and code (in fact, it has been implemented in 3 lines of PERL, 4 lines of C, etc.). Its speed and memory efficiency are amazing when you compare to any of the provably secure PRGs we built. Unfortunately, you can't prove much about it, although it has been extensively studied.

1.3 DES

DES is well-described in many public sources (in particular, the standard itself [NIS77] is quite clear and is on-line at `http://www.itl.nist.gob/fipspubs/`), or see p. 218 of the textbook); hence the description is omitted here.

1.4 Pseudorandom Functions

We now turn to formal definitions again. We already know what pseudorandom generators are. We define pseudorandom functions.

A pseudorandom function family (PRF) is a collection of efficiently computable functions such that choosing a random function from the family is as good (with respect to polynomial-time distinguishers) as choosing a truly random function. The fact that these things exist is remarkable, since most function are not even computable (let alone efficiently computable), and yet a tiny (in comparison) subset of efficiently computable collection can be as good as the collection of all functions.

Definition 1. Let $i(k)$ and $o(k)$ be the input and output lengths, respectively, for security parameter k. A family of functions $\{F_s\}_{s \in S}$ is pseudorandom if

- There exists a polynomial-time algorithm $\mathrm{Gen}(1^k)$ that outputs s on input 1^k, such that F_s maps $\{0,1\}^{i(k)}$ to $\{0,1\}^{o(k)}$.
- There exists a polynomial-time algorithm that outputs $F_s(x)$ given s and x.
- For every probabilistic polynomial time oracle machine $A^?$ there exists a negligible function η such that
$$\left| \Pr_{f \text{ random function } \{0,1\}^{i(k)} \to \{0,1\}^{o(k)}} [A^f(1^k) \to 1] - \Pr_{s \leftarrow \mathrm{Gen}(1^k)} [A^{F_s}(1^k) \to 1] \right| \le \eta(k).$$

The value s is usually called the *seed*.

The above definition and the following construction are due to Goldreich, Goldwasser and Micali [GGM86]. While many believe that things like DES approximates PRFs, it's good to convince ourselves that such things exist by building them out of reasonable assumptions.

To build a PRF, let G be a length-doubling PRG: given s of length k, G outputs y of length $2k$. Let $G_0(s)$ be the first k bits of y, and $G_1(s)$ be the last k bits of y. Define $F_s(x)$, for a k-bit seed and k-bit input $x = x_1 x_2 \dots x_k$, where x_i is a bit, to be $F_s(x) = G_{x_k}(G_{x_{k-1}}(G_{k_{k-2}}(\dots G_{x_1}(s))))$. In other words, build a tree of height k defined as follows:

- The seed value s is contained at the root
- The left child of a node containing α contains $G_0(\alpha)$
- The right child of a node containing α contains $G_1(\alpha)$

Then on input x output the value stored at the leaf number x.

The proof that it's a PRF is done by a hybrid argument on the levels of the tree; we omit it here.

There are more efficient constructions of PRFs, e.g., [NR97, NRR01], but we will not study them for lack of time.

One way to think of a PRF F_s is to think of it as a PRG on the seed s with exponentially long output (simply concatenate all output values for all possible x's into one). Since no one can read or write exponentially long values, we simply give the distinguisher random access to the output (rather than sequential access given in the case of PRG).

PRFs vs. Random Oracles We note while PRFs are indistinguishable from truly random functions, they cannot be used as public random oracles. This is because indistinguishability holds only with respect to *oracle access*: i.e., the adversary is allowed to ask questions x and receive answers $F_s(x)$, but does not know how to compute F_s herself. If the adversary knows the seed s, then the function no longer looks random. When we needed random oracles for signature schemes, we needed *everyone* to be able to evaluate the random oracle, so we would have to make the seed public, so it would no longer look random.

1.5 Pseudorandom Permutations

Pseudorandom permutations (PRPs) are defined the same way as pseudorandom functions, with the following additional properties: $i(k) = o(k)$, F_s is a permutation, and there exists an algorithm that, given s and y, computes $x = F_s^{-1}(y)$. Since block cipher satisfy these additional properties and appear to have pseudorandomness properties, many believe that it is appropriate to model secure block ciphers as PRPs.

However, we should convince ourselves that such things really exist based on some plausible assumptions (just like we did for PRFs). First, how can it be that a *permutation* looks indistinguishable from a random *function*: a random functions has a lot of collisions. Well, since our distinguisher is only allowed polynomially many queries, a random permutation and a random function actually look the same, since the likelihood of a collision for polynomially many inputs is negligible.

To build PRPs, we start with PRFs, and apply the following idea (called the Feistel permutation since the time when DES was designed [Fei73, FNS75], but actually due to Notz and Smith according to [Cop00]).

Let F_{s_1} be a PRF mapping k bits to k bits. Let x be a $2k$-bit quantity; write it as $x = (L, R)$, where L and R are k bits each; define $\Psi_{s_1}(L, R) = (R, S)$ where $S = L \oplus F_{s_1}(R)$. Note that $\Psi_{s_1}(x)$ is a permutation that is easy to invert: simply apply F_{s_1} to the first part of the output, and exclusive-or it with the second. Thus, we built a permutation on $2k$ bits out of a PRF on k bits.

Is this permutation pseudorandom? Of course not: half the output bits are the same as half of the input bits. Well, what if we compose (chain) such permutations: pick another seed s_2 and apply Ψ_{s_2} to (R, S) to get (S, T), where $T = R \oplus F_{s_2}(S)$. The result—$\Psi_{s_1} \circ \Psi_{s_2}$—is of course a permutation, but is still not pseudorandom (consider what happens to S when the adversary asks two queries with the same R but different L). Let's try again: pick another seed s_3 and apply Ψ_{s_3} to (S, T) to get (T, V), where $V = S \oplus \Psi_{s_3}(T)$. Turns out that this is pseudorandom.

To be precise, the result of Luby and Rackoff [LR88] states the following. Given a PRF $\{F_s\}_{s\in S}$ where $i(k) = o(k) = k$, consider the following family of permutations. To generate a seed, run $\text{Gen}(1^k)$ three times to get seeds s_1, s_2, s_3. Then let $\Phi_{s_1 s_2 s_3}(L, R) = \Psi_{s_3}(\Psi_{s_2}(\Psi_{s_1}(L, R))) = (T, V)$, where

$$\begin{aligned} S &= L \oplus F_{s_1}(R) \\ T &= R \oplus F_{s_2}(S) \\ V &= S \oplus F_{s_3}(T). \end{aligned}$$

Then $\{\Phi_{s_1 s_2 s_3}\}$ is a pseudorandom permutation family.

Incidentally, since we are speaking of permutations, one can consider what happens if the adversary is given access to the inverse direction (i.e., the adversary has to distinguish a random permutation from a pseudorandom one when given access to both forward and inverse direction). A permutation family that passes this test is called *super pseudorandom*. Turns out that the above construction fails (with only three queries from the adversary!). However, adding one more round Ψ_{s_4} makes it secure against this stronger adversary.

Remarkably, the Feistel permutation idea was first used to turn functions into permutations well before modern notions of pseudorandomness. Namely, the round function of Lucifer (the precursor cipher to DES) was a permutation, so Lucifer was invertible. But the round function of DES was not a permutation. To make DES invertible, two of the DES designers, Notz and Smith, proposed to split the input into two halves, apply the function to only one half, and use the XOR operation. This is exactly what each of the 16 rounds of DES does. It is quite surprising that this idea actually works provably to turn to pseudorandom functions into pseudorandom permutations.

1.6 Symmetric Encryption

1.6.1 Definition

Syntactically, secure symmetric encryption is defined similarly to secure public-key encryption, except that there is a single key.

Definition 2. A *symmetric cryptosystem* is a triple of polynomial-time algorithms $(\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$. $\mathrm{Gen}(1^k)$ is a (randomized) key-generation algorithm that outputs a key K when given a security parameter k as input. Enc is a (randomized) encryption algorithm that, on input K and message m, outputs ciphertext c. Dec is a (usually deterministic) decryption algorithm, that, on input K and c, outputs m. For a key K, a cryptosystem has to specify a set of allowed messages M_K (ultimately the goal will be to have M be all binary strings regardless of K; however, we have to allow for less general encryption schemes at first). We require that the following holds: if K is produced by $\mathrm{Gen}(1^k)$, then for all $m \in M_K$, $m = \mathrm{Dec}_K(\mathrm{Enc}_K(m))$ (this requirement can be relaxed to say "with probability $1 - \eta(k)$").

When we defined secure public-key encryption, the adversary Eve had to distinguish between encryptions of two messages. While in real life Eve could have seen encryptions of other messages, too, they could not have been helpful, because she could have produced those encryptions herself (all she needs to produce encryptions is the public key, which she has). This is not the case for symmetric encryption: perhaps she may gain something by seeing encryptions of other messages, since she can't produce them herself. Moreover, in real life, she may well have access to some ciphertexts (and perhaps even know, or be able to influence, their corresponding plaintexts). So to get a meaningful definition, we should at least give Eve oracles access to the encryption oracle, so that she can input plaintexts and see corresponding ciphertexts. This is called a "chosen-plaintext attack" (CPA). Note that in the case of public-key encryption, Eve gets the public key, and hence gets CPA automatically, without any oracles.

We define security, as usual, using two experiments. The adversary in these experiments is a distinguisher D that runs in two stages and is allowed to keep state between stages.

$\texttt{exp-}m_0(\mathrm{k})$
1. $K \leftarrow \mathrm{Gen}(1^k)$
2. $(m_0, m_1) \leftarrow D^{\mathrm{Enc}_K(\cdot)}(1^k)$; if $|m_0| \neq |m_1|$, abort
3. $c \leftarrow \mathrm{Enc}_K(m_0)$
4. Output $D^{\mathrm{Enc}_K(\cdot)}(c)$

The second experiment $\texttt{exp-}m_1$ is the same, except in line 3, which changes to $c \leftarrow \mathrm{Enc}_K(m_1)$.

Definition 3. A symmetric cryptosystem is *polynomially-secure under CPA* if for all polynomial time $D^?$ there exists a negligible function $\eta(k)$ such that

$$|\Pr[\texttt{exp-}m_0(k) \text{ outputs } 1] - \Pr[\texttt{exp-}m_1(k) \text{ outputs } 1]| \leq \eta(k).$$

Equivalently, we could have considered a random experiment where a bit b gets chosen random, m_b gets encrypted, and D has to output a guess g for the bit b. Security requires that the probability $b = g$ be negligibly greater than $1/2$.

1.6.2 Left-or-Right Definition

An alternative definition is to give Eve oracle access to the encryption oracle with one twist: she has to give *two* messages (of the same length) for each query, and we will pick which one gets encrypted in each pair (the choice will be the same for all pairs, but unknown to Eve). She will have to decide which of the two messages we are consistently choosing. Notice that Eve is allowed to have two messages in a pair be the same message m, the ensuring that she will see an encryption of m. (However, if her goal is to distinguish, then she will have to give different messages at some point.) This definition was proposed by Bellare, Desai, Jokipii and Rogaway [BDJR97]; its equivalence to other notions (including the above definition and a version of semantic security for symmetric encryption) is proven in [BDJR97], as well.

Definition 4 ([BDJR97]). A symmetric cryptosystem is *polynomially-secure* against adaptive chosen-plaintext attack (CPA) if for all polynomial time $E^?$ there exists a negligible function $\eta(k)$ such that

$$\left| \Pr_{K \leftarrow \mathrm{Gen}(1^k)} [E^{\mathrm{Enc}_K(LR(\cdot,\cdot,0))}(1^k) \to 1] - \Pr_{K \leftarrow \mathrm{Gen}(1^k)} [E^{\mathrm{Enc}_K(LR(\cdot,\cdot,1))} \to 1] \right| \leq \eta(k),$$

where $LR(m_0, m_1, b) \overset{def}{=} m_b$ if $|m_0| = |m_1|$ and ϵ otherwise. The probabilities above are taken over the random choices made by Gen to generate K, by Enc in answering oracle queries, and by E.

1.6.3 Constructions

Observe that the adversary is allowed to query its encryption oracle to get encryptions of both m_0 and m_1. Thus, secure symmetric encryption must be probabilistic or stateful, to ensure that, at the very least, encryption of m_0 comes out different each time (otherwise, it would be easy to distinguish: simply check if challenge ciphertext c matches the output of the oracle on m_0).

One-Time Pad We'll start from a construction we already know: the one-time pad. Let $\mathrm{Gen}(1^k)$ simply output a k-bit random key K; let $M_K = \{0,1\}^{|K|}$. To encrypt m, let $c = m \oplus K$; to decrypt c, let $m = c \oplus K$. If $D^?$ is not allowed any queries to its oracle, then $D^?$ cannot distinguish: the probability of receiving any ciphertext c if m_0 gets encrypted is the same as the probability of receiving it if m_1 gets encrypted (by perfect secrecy). Hence, the view of $D^?$ is the same, and it cannot distinguish.

However, clearly, if D is allowed to query the oracle, then it can get the key K. That's why it's called the *one-time* pad, after all.

We can extend one-time pad to l-time pad, by outputting K of length lk, and using subsequent portions of it for each query (thus, this encryption scheme would be *stateful*, because Enc and Dec would be required to know what message number they are one; Dec could be made stateless if Enc output the message number together with the ciphertext). This would be secure up to $l-1$ queries, but not beyond.

Stream Cipher Encryption Now modify the l-time pad by having Gen output K as a seed for a PRG G with unlimited output length (all PRGs that we studied have unlimited output length, anyway). Then keep track of how many bits you've encrypted so far. If you've encrypted t bits so far and need to encrypt n bits now, simply exclusive-or the message with the bits $t+1, t+2, \ldots, t+n$ of $G(K)$ to get c and output $(t+1, c)$. Update $t \leftarrow t+n$. This give secure stateful encryption (only the encryptor needs to keep state, since the decryptor gets $t+1$). We give no formal proof here, but it simply formalizes the following intuition: a pseudorandom pad better be as good as a truly random pad, because otherwise we could build a distinguisher for the PRG. But a truly random pad of unbounded length is secure, as we said above.

PRF-based encryption Let Gen output a seed s for a PRF F_s with k-bit inputs and k-bit outputs. Then to encrypt a k-bit message m, choose a random x and output $c = (x, m \oplus F_s(x))$. Clearly, if F_s were a truly random function, this would be secure (since it's essentially a new one-time pad for every message, unless you hit the same x twice, which is extremely unlikely). By the same reasoning as for stream ciphers, this ought to be secure for PRFs, since otherwise you could distinguish PRFs from truly random functions.

To encrypt longer messages, e.g., a message with l blocks of k bits each, $m = m_1 \dots m_l$, pick a random x and output $c = (x, m_1 \oplus F_s(x), m_2 \oplus F_s(x+1), m_3 \oplus F_s(x+2), \dots, m_l \oplus F_s(x+l-1))$. This encryption procedure is known as "counter encryption mode" for block ciphers, and is often abbreviated CTR. Note that instead of picking a random x each time, one can make encryption stateful and simply use $x + l$ next time as a starting point.

The insecure ECB mode Note that it would be insecure to simply split up the input message into l blocks and pass each block through F_s. This encryption is deterministic, and as we already said, deterministic encryption cannot be secure. This mode of encryption is known as ECB [NIS80] for "electronic codebook." The name goes back to the times when people carried around codebooks of substitutions rules — ECB mode simply substitutes blocks of output for blocks of input in a deterministic manner.

CBC-mode encryption A common way to encrypt a long message is the following, known as "cipher block chaining" or CBC (also specified in [NIS80]). Let F_s be a PRF with k-bit inputs and k-bit outputs. To encrypt m with l blocks of k bits each, $m = m_1 \dots m_l$, pick a random value y_0 (known as "initialization vector" or "IV"), and let $y_1 = F_s(m_1 \oplus y_0)$, $y_2 = F_s(m_2 \oplus y_1)$, $\dots$, $y_l = F_s(m_l \oplus y_{l-1})$. The ciphertext is $y_0 y_1 \dots y_l$. To decrypt it, compute $m_1 = F_s^{-1}(y_1) \oplus y_0, \dots, m_l = F_s^{-1}(y_l) \oplus y_{l-1}$. Note that decryption can be done in parallel, despite the chaining.

Observe that decryption requires us to be able to compute F_s^{-1}, which is not specified in the definition of PRF. In fact, F_s^{-1} might not even be well-defined, because F_s may not be one-to-one. Thus, to use CBC mode, one needs pseudorandom permutations, as opposed to just pseudorandom functions. In fact, one can prove [BDJR97] that it is secure if F_s is a pseudorandom permutation.

1.7 Message Authentication Codes

1.7.1 Definition

Message Authentication Codes (MACs) are the symmetric equivalent of signatures. The definition is essentially the same, except that to have a convincingly strong definition, we need to give the adversary the power to query not only the signing oracle, but also the verifying oracle, because (unlike in the public-key case) the adversary cannot verify on its own. Also, a symmetric signature is traditionally called a "tag."

Formally, a MAC is a triple of probabilistic polynomial-time algorithms (Gen, Tag, Ver). The key generation algorithm Gen outputs K when given 1^k as input. The tagging algorithm Tag takes K and m as input, and outputs a tag σ. The verification algorithm Ver takes K, m, σ as input and outputs 1 or 0 (or true/false, valid/invalid, etc.). We require that tags produced by Tag verify as correct by Ver: if $K \leftarrow \mathrm{Gen}(1^k)$, then for all m, $\mathrm{Ver}_K(m, \mathrm{Tag}_K(m)) = 1$ (perhaps with probability $1 - \eta(k)$). We may also restrict the message space to some set M, and instead saying "for all m," say "for all $m \in M$."

Security is defined in terms of the following experiment.

`exp-forge`(k)

1. $K \leftarrow \mathrm{Gen}(1^k)$
2. $(m, \sigma) \leftarrow E^{\mathrm{Tag}_K(\cdot), \mathrm{Ver}_K(\cdot,\cdot)}(1^k)$
3. If m was not queried by E to its signing oracle and $\mathrm{Ver}_K(m, \sigma) = 1$, output 1. Else output 0.

Definition 5. We say that a MAC is secure (existentially unforgeable under an adaptive chosen-message attack) if for all probabilistic polynomial time $E^?$ there exists a negligible function η such that $\Pr[\texttt{exp-forge}(k) \to 1] \leq \eta(k)$.

1.7.2 Constructions

PRFs A PRF is a MAC. That is, if $\{F_s\}_{s\in S}$ is a PRF with some sufficiently long output length (precisely, if $i(k)$ is polynomial in k), then a MAC key is simply a PRF seed s, and to tag a message m, simply compute $\sigma = F_s(m)$; to verify, check if this holds. This works for message of length $o(k)$ (we give no proof of security here, but it's quite simple). But what to do for longer messages?

Hashing One idea is to use collision-resistant hashing to hash the message down to a shorter one, just like we did for digital signatures. However, collision-resistant hashing is an overkill: it requires the hash function to be collision-resistant even when the key is known to the adversary. This is necessary in the public-key case (when the verifier must know the hash key), but not in the symmetric case, when the hash key can be kept secret. Thus, we need a much simpler primitive, known as "universal hashing."

Combining a universal hash function with any secure MAC for short messages (such as PRF) gives you a secure MAC for long messages.

Definition 6. Let $i(k)$ and $o(k)$ be the input and output lengths, respectively, for security parameter k. A family of functions $\{H_i\}_{i\in I}$ is a universal hash family if:

- There exists a polynomial-time algorithm $\text{Gen}(1^k)$ that outputs i on input 1^k, such that H_i maps $\{0,1\}^{i(k)}$ to $\{0,1\}^{o(k)}$.
- There exists a polynomial-time algorithm that outputs $H_i(x)$ given i and x.
- There exists a negligible function η such that for all k, for all $x_1, x_2 \in \{0,1\}^{i(k)}$,

$$\Pr_{i\leftarrow \text{Gen}(1^k)}[H_i(x_1) = H_i(x_2)] \leq \eta(k).$$

Unfortunately, we have no time to spend on examples of universal hashing; we will simply say that it's very easy construct, and *no cryptography is needed* (i.e., one need not make complexity-theoretic assumptions). Simple linear algebra works: for example, if $i(k) = m$ and $o(k) = n$, then then letting H_i be a random linear transform from $GF(2)^m$ to $GF(2)^n$ works.

CBC MAC Probably the most popular MAC in practice the following: to MAC a message, CBC-encrypt it with IV$= 0$, and output the very last block as the tag.

Formally, let $\{F_s\}_{s\in S}$ be a PRF. Key generation algorithm just selects the seed s, with $F_s : \{0,1\}^l \to \{0,1\}^l$. To tag a message m consisting of n l-bit blocks $m = m_1m_2\dots m_n$, compute and output y_n, where $y_i = F_s(m_i \oplus y_{i-1})$ and $y_0 = 0^l$. To verify, repeat the computation and check if the tag matches.

Turns out this is secure only for fixed-length messages: i.e., if the adversary's queries and the eventual forgery have to be the same length. However, if the adversary is allowed to change the length, then it's insecure. This can be fixed by prepending (but, surprisingly, not appending!) the length to the message; or by simply passing y_n through a PRF with an independent seed s' (thus, the key becomes $K = (s, s')$). See [BKR94, PR00] for more on CBC MAC.

Be careful, however, not to assume that just because CBC MAC is secure, CBC encryption also provides authenticity. It does not.

1.8 Combining Authentication and Encryption

Often you want to send a message that is both secret and authentic. In the symmetric-key setting it turns out that, if done properly, this will increase the security of the encryption itself. Namely, if you encrypt the message using any CPA-secure encryption scheme, and then MAC the ciphertext using any secure[1] MAC, then you get CCA2-secure encryption and authenticity. Thus, by simply adding authentication, you increase the strength of your encryption from chosen-plaintext secure to chosen-ciphertext secure. (Note that the order of MAC and encrypt matters; see [BN00] for more.)

Note that doing both encryption and MACing is a bit expensive: e.g., if you use CBC MAC, it's twice as expensive as encryption itself. There is work on encryption modes that provide both encryption and authentication at the same cost as just encryption (see, e.g., the OCB mode of [RBBK01]).

The story is more complicated in the public-key world, because combining encryption and authentication involves using your own secret key to sign the message, and then encrypting both the message and the signature with the public key of the recipient. The keys are different, and procedures for encrypting and signing are very different. In particular, security does not necessarily automatically increase from CPA to CCA as it does in the symmetric setting. Nonetheless, there is interesting work there as well. Combinations of signatures and encryption are often called "signcryption." See, e.g., [ADR02] for more.

References

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*. Springer-Verlag, 28 April–2 May 2002.

[BDJR97] Mihir Bellare, Anand Desai, E. Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science* [IEE97].

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. In Yvo G. Desmedt, editor, *Advances in Cryptology—CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer-Verlag, 21–25 August 1994.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, Kyoto, Japan, 3–7 December 2000. Springer-Verlag.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology—EUROCRYPT 94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995, 9–12 May 1994. Revised version available from `http://www-cse.ucsd.edu/users/mihir/`.

[Cop00] Don Coppersmith. Invited lecture: The development of DES. In Mihir Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*. Springer-Verlag, 20–24 August 2000.

[Fei73] H. Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, May 1973.

[1] You need a MAC with a slightly stronger security property: it should be hard not only to forge a tag on a new message, but also to forge a new tag on an old message. All the MACs we discussed satisfy this property, because there is only one correct tag for each message.

[FNS75] H. Feistel, W. A. Notz, and J. L. Smith. Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545–1554, 1975.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[IEE97] IEEE. *38th Annual Symposium on Foundations of Computer Science*, Miami Beach, Florida, 20–22 October 1997.

[LR88] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, April 1988.

[NIS77] FIPS publication 46: Data encryption standard, 1977. Available from `http://www.itl.nist.gov/fipspubs/`.

[NIS80] FIPS publication 81: DES modes of operation, 1980. Available from `http://www.itl.nist.gov/fipspubs/`.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science* [IEE97], pages 458–467.

[NRR01] Moni Naor, Omer Reingold, and Alon Rosen. Pseudo-random functions and factoring. *Electronic Colloquium on Computational Complexity (ECCC)*, TR01-064, 2001.

[PR00] Erez Petrank and Charles Rackoff. CBC MAC for real-time data sources. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(3):315–338, 2000.

[RBBK01] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Eighth ACM Conference on Computer and Communication Security*, pages 196–205. ACM, November 5–8 2001. Full version available from `http://www.cs.ucsdavis.edu/~rogaway`.

[Riv87] Ronald L. Rivest. The RC4 encryption algorithm. Trade secret of RSA Data Security, Inc.; leaked and subsequently published in [Sch95], 1987.

[RSA02] PKCS #1: RSA encryption standard. Version 2.1, June 2002. Available from `http://www.rsaisecurity.com/rsalabs/pkcs/`.

[Sch95] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, second edition, 1995.