Symmetric Encryption: PRFs, Counter mode, and ChaCha20

Notes by Henry Corrigan-Gibbs

MIT - 6.5610
Lecture 3 (February 10, 2025)

Warning: This document is a rough draft, so it may contain bugs. Please feel free to email me with corrections.

Logistics

- Due Friday: Pset 1
- Due Friday: Post a project idea on Piazza

Outline

- Reminder: Definition of a PRF
- Reminder: CPA-Secure Encryption
- ChaCha20

Why study the design of symmetric-key primitives?

This week, we are going to study symmetric-key cryptographic primitives.

- These are arguably the most important primitives in the practice of crypto.
  They are used in essentially every device you own that uses encryption.
- The goal is **NOT** for you to write your own implementations or design your own ciphers. The goal is for you to have a deeper understanding of how encryption systems work in practice.

Review: Pseudorandom functions

Definition

This discussion of PRFs is copied almost verbatim from our 6.1600 lecture notes.

Syntax A *pseudorandom function* (PRF) is defined over a keyspace $\mathcal{K}$, and input space $\mathcal{X}$ and output space $\mathcal{Y}$ is a function $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. For concreteness, we can think of $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$.

If we want to be completely formal, we parameterize the key space (also possibly the input/output spaces) by a security parameter λ. We then write $F = \{F_\lambda\}_{\lambda \in \mathbb{N}}$, where, for each $\lambda \in \mathbb{N}$, the function F_λ has type $F_\lambda\colon \mathcal{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$.

Security Informally, a pseudorandom function must "look like" a random function in the sense that for secret $k \xleftarrow{R} \mathcal{K}$, it is infeasible to distinguish $F(k,\cdot)$ from a truly random function $f\colon \mathcal{X} \to \mathcal{Y}$.

The style of definition here follows the (free!) Boneh-Shoup textbook. Check it out for much more detail on these topics.

Formally, we define PRF security using a game:

Definition 1 (PRF Security Game)**.** The game is parameterized by a PRF $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, and adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$.

- The challenger samples a key $k \xleftarrow{R} \mathcal{K}$.
- If $b = 0$, the challenger sets $f(\cdot) := F(k,\cdot)$.
- If $b = 1$, the challenger sets $f(\cdot) \xleftarrow{R} \mathsf{Funs}[\mathcal{X},\mathcal{Y}]$.
- Then for $i = 1,2,\dots$ (polynomially many times):
  - The adversary $\mathcal{A}$ sends the challenger a value $x_i \in \mathcal{X}$.
  - The challenger responds with $y_i \leftarrow f(x_i) \in \mathcal{Y}$.
- The adversary outputs a bit $\hat{b}$.

Here, Funs is the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$.

For $b \in \{0,1\}$, let W_b denote the probability that some adversary $\mathcal{A}$ outputs bit "1" in the PRF security game parameterized with bit b. Then define the PRF advantage of $\mathcal{A}$ at attacking F as:

$$\mathsf{PRFAdv}[\mathcal{A},F] := \left|\Pr[W_0] - \Pr[W_1]\right|.$$

Definition 2 (Pseudorandom function)**.** A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a pseudorandom function if, for all "efficient" algorithms $\mathcal{A}$,

$$\mathsf{PRFAdv}[\mathcal{A}, F](\lambda) \leq \text{"negligible."}$$

The adversary that guesses a random bit has advantage 0. This definition asserts that no efficient adversary can do much better than that.

There are two views on this definition:

- **In theory**, we parameterize everything by a security parameter λ, which you can think of the length of the secret PRF key. We require the PRF algorithm and adversary to run in time polynomial in λ, and we define "negligible" to be a function that is negligible in λ.

- **In practice**, we fix keysize to some constant—often 128 or 256 bits. We require the PRF algorithm to be "fast enough," we aim to defend against attackers than run in time (say) 2^{80}, and we say define "negligible" to be some small constant, such as 2^{-64}.

Reminder If $\mathsf{P} = \mathsf{NP}$ then pseudorandom functions do not exist.

To explain: A poly-time algorithm for NP problems can efficiently find a satisfying assignment to a Boolean circuit. To use a circuit-SAT solver to break a PRF, an attacker can query its oracle in the PRF security game on a few points $x_1, x_2, \dots$, getting answers $y_1, y_2, \dots$. The attacker then uses it's circuit-SAT solver to find a PRF key k that is a satisfying assignment to the circuit

$$C(k) = \{y_1 = F(k, x_1) \wedge y_2 = F(k, x_2) \wedge \dots\}.$$

If such a key k exists, then almost certainly the attacker is in the PRF world. Otherwise not.

This discussion implies that the existence of secure a pseudorandom functions implies that $\mathsf{P} \neq \mathsf{NP}$ at least.

Symmetric-key assumptions

Some of crypto is based on "nice" assumptions. For example, the Rabin cryptosystem is based on the hardness of factoring. This is a "win-win" situation: either we have a secure cryptosystem, *or* we get a factoring algorithm (which would be exciting).

We could base our block ciphers on "nice" assumptions, such as the assumption that factoring is hard, but the resulting cryptosystems would be too slow. Instead, we design symmetric-key cryptosystems based on ad-hoc assumptions, which cryptanalysts then spend many years trying to break.

For example, we just assume that AES is a secure block cipher—there is no clean mathematical assumption to which we can relate its security. At the same time, most cryptographers I know have much more confidence that AES is a secure PRF than that factoring or discrete log is hard.

The plan for evaluating the security of new primitives is:

- Try to break the new primitive with all known attacks.
- Run competitions and to get researchers to break each other's cryptosystems.
- After a design has withstood a few years of scrutiny, assume that it's good enough.

Having said that, the difficulty of cipher design at this point often is not security, but getting good performance on all sorts of different hardware.

Encryption

Review: Security against chosen-plaintext attacks (CPA security)

An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is a pair of algorithms defined over a common key space $\mathcal{K}$, message space $\mathcal{M}$, and ciphertext space $\mathcal{C}$ with syntax: $\mathsf{Enc}\colon \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ and $\mathsf{Dec}\colon \mathcal{K} \times \mathcal{C} \to \mathcal{M}$.

To be useful an encryption scheme must be **correct** and **secure**.

chosen-plaintext attacks, which I will not define formally. CPA security is a relatively weak definition of security. In practice we use encryption systems that satisfy "authenticated encryption"—a much stronger definition of security. The 6.1600 lecture notes have lots of details on CPA security and authenticated encryption, so look there for more information. Suffice it to say this the CPA-secure encryption scheme is a useful building block but is NOT one that you would ever use in practice to encrypt messages.

Definition 3 (CPA Security game)**.** The game is parameterized by an encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ over message space $\mathcal{M}$, key space $\mathcal{K}$, and a bit $b \in \{0,1\}$:

- The challenger samples $k \xleftarrow{R} \mathcal{K}$.
- As many times as the adversary wants:
  - The adversary sends the challenger messages $m_{i0}, m_{i1} \in \mathcal{M}$. (We require $|m_0^*| = |m_1^*|$.)
  - The challenger replies with $c_i \leftarrow \mathsf{Enc}(k, m_{ib})$.
- The adversary outputs a value $b' \in \{0,1\}$.

Standard encryption systems do not hide the length of the message being encrypted. So, if the message space $\mathcal{M}$ contains messages of different lengths, our security definition requires the adversary to distinguish the encryption of two messages of the *same* length.

For an adversary $\mathcal{A}$ and $b \in \{0,1\}$, let W_b be the event that the adversary outputs "1" in the above game. Then we say that an encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ is **CPA-secure** if for all "efficient" adversaries $\mathcal{A}$, we have

$$\mathsf{CPAAdv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]| \leq \text{“negligible.”}$$

Encrypting with a pseudorandom function: Counter mode

Last time, Yael demonstrated how to encrypt messages using a pseudorandom function. Say that we have a pseudorandom function $F\colon \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$. The encryption scheme is is called counter- or CTR-mode encryption. The encryption scheme is parameterized by a value $\ell \in \mathbb{N}$, which determines the length of messages that the scheme can encrypt.

Then the encryption scheme is defined over:

- Key space $\mathcal{K}$
- Message space $\{0,1\}^{n\ell}$
- Ciphertext space $\{0,1\}^{n\ell}$

The algorithms are:

$\mathsf{Enc}(k, (m_1, \dots, m_\ell))$:

- Sample $r \xleftarrow{R} \{0,1\}^n$. *// r is called a "nonce"*
- Output $(r, F(k,r) \oplus m_1, F(k, r+1) \oplus m_2, \dots, F(k, r+\ell-1) \oplus m_\ell)$.

$\mathsf{Dec}(k, (r, c_1, \dots, c_\ell))$:

- Output $F(k,r) \oplus c_1, F(k, r+1) \oplus c_2, \dots, F(k, r+\ell-1) \oplus c_\ell)$.

The sort of security statement you can prove about counter-mode encryption is that for all adversaries $\mathcal{A}$ making at most T encryption queries of length-ℓ messages, there is a PRF adversary $\mathcal{B}$ (running in roughly the same time as $\mathcal{A}$) such that

See the Boneh-Shoup book, Theorem 5.3, for a formal proof of this.

$$\mathsf{CPAAdv}[A, \mathcal{E}] \leq 2 \cdot \mathsf{PRFAdv}[\mathcal{B}, F] + \frac{2T^2\ell}{2^n}.$$

Thus, if we instantiate counter-mode encryption with a secure PRF, we get a CPA-secure encryption scheme.

The idea of the proof is to first imagine swapping out the PRF with a truly random function. If the PRF is secure, then making this switch cannot change the adversary's advantage by too much. Then, provided that the encryptor never re-uses a nonce, the adversary is just seeing messages XORd with truly random values.

If we believe the underlying PRF to be secure, then we believe that $\mathsf{PRFAdv}[\mathcal{B}, F]$ to be "negligible" for all efficient algorithms $\mathcal{B}$. Then as long as $T^2\ell \ll 2^n$, we expect the encryption scheme to be CPA secure.

What happens if $T^2\ell$ is roughly equal to 2^n? If you think about it for a bit, you may be able to see that the encryption scheme is actually broken: there is an attack on CPA security!

Parallel implementation. Counter-mode encryption lends itself to parallel implementation on multicore machines: it is possible for an encryptor or decryptor to decrypt many blocks of a message in parallel. Since modern machines often have 10+ cores, exploit parallelism is critical for modern cryptosystems. (Thirty years ago, multicore architectures were much less common and so parallelism was not as much of a concern.)

Stateful counter mode. It is not actually important that the encryption nonce r be random—only that the encryption scheme never reuse the value $F(k, r)$ to encrypt two distinct messages. In applications in which the sender and receiver can share state, they can start with the nonce $r = 0$ and can increment it appropriately after each message sent. This avoids the need to send r along with the ciphertext, at the cost of having to keep synchronized state.

Why not build an encryption scheme directly? Here, we started with a small primitive—a pseudorandom function with a fixed input and output size—and used it to construct a bigger one—an encryption system for long messages. Why not just construct an encryption scheme for large messages directly? Why start with a pseudorandom function?

A theoretical reason is that we want to understand what the minimal assumption we need to make to construct a cryptographic object.

A practical reason is that cryptanalysis is extremely time consuming and costly. Being able to build a large array of tools from a very small set of primitives lets the cryptanalysts focus their efforts on just these few very important primitives.

STRETCH BREAK?

ChaCha20: Example of a PRF used for counter-mode encryption

ChaCha20 is a stream cipher that is used in the TLS protocol that secures your HTTP connections. ChaCha20 encrypts messages essentially by constructing a pseudorandom function and then using it in counter mode.

The ChaCha20 PRF uses a 256-bit key and a 128-bit input and outputs a 512-bit value:

$$F_{\mathsf{chacha}} \colon \{0,1\}^{256} \times \{0,1\}^{128} \to \{0,1\}^{512}.$$

Let's see how this particular PRF is constructed. Like many symmetric-key primitives, ChaCha20 is itself built from a lower-level primitive—in this case a public permutation on $\{0,1\}^{512}$. If were to model the public permutation as a truly random object, we could *prove* (in this idealized model) that ChaCha20 is a secure PRF. The "leap of faith" that we need to make is that of course the public permutation in the ChaCha20 design is not a truly random object (that would take roughly 2^{512} bits to describe)—it has a very short and simple implementation.

The PRF defines a function $\mathsf{pad}(k, x)$ that maps the key k, the input x, and 128-bits worth of constants, to a 4×4 matrix of 32-bit values. The PRF construction also defines a permutation $\Pi \colon \{0,1\}^{512} \to \{0,1\}^{512}$.

The constants spell `expand 32-byte k`. This an example of a "nothing-up-my-sleeve" number.

The PRF output is then:

$$F_{\mathsf{chacha}}(k, x) := \mathsf{pad}(k, x) \oplus \Pi(\mathsf{pad}(k, x)) \quad \in \{0,1\}^{512}.$$

The permutation Π. The only thing left to specify is the permutation Π. The core is a function $\mathsf{quarterRound}$ that takes as input 4 32-bit values and outputs 4 32-bit values. The function for Salsa (slightly simpler than ChaCha) is:

```
quarterRound(a,b,c,d):
  b ^= (a + d) <<<  7;
  c ^= (b + a) <<<  9;
  d ^= (c + b) <<< 13;
  a ^= (d + c) <<< 18;
```

where <<< is a bitwise left-rotate operation.

ChaCha20 is called an ARX cipher, since its core consists of Add, Rotate, and XOR operations.

Viewing the input to Π as a 4×4 matrix of 32-bit values, the permutation Π applies the quarter-round function to the columns of the matrix:

```
 0 |  1 |  2 |  3
 4 |  5 |  6 |  7
 8 |  9 | 10 | 11
12 | 13 | 14 | 15
```

and then to the shifted columns:

```
 0 |  1 |  2 |  3
 5 |  6 |  7 |  4
10 | 11 |  8 |  9
15 | 12 | 13 | 14
```

The permutation iterates this function 10 times.

Notice that an implementation can apply the 4 instances of the quarter-round function in parallel. Moreover, these are SIMD computations, for which many CPUs have special instructions.

In addition, there are no secret- or data-dependent memory accesses. This is critical for security to avoid timing and cache attacks, in which an attacker learns bits of information about a victim's secrets via how long memory accesses take to complete.

References