Course Introduction

Notes by Henry Corrigan-Gibbs

MIT - 6.5610

Lecture 2 (February 7, 2024)

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

Logistics

- You should have received a group assignment

Outline

- Reminder: Definition of a PRF
- DES
- **Stretch break**
- AES

Quantum vs. post-quantum crypto

One important distinction that I did not mention last lecture is the distinction between *post-quantum cryptography* and *quantum cryptography*.

When we are talking about quantum algorithms, there are actually three settings we can think about:

	Honest party	Adversary
Classical crypto	Classical	Classical
Post-quantum crypto (us)	Classical	Quantum
Quantum crypto (6.S895)	Quantum	Quantum

It does not make much sense to think about a quantum honest party and classical adversary, because then the honest party is more powerful than the adversary.

In the setting of quantum crypto, we typically think of the honest party and adversaries communicating over *quantum* channels.

The new NIST standards are on post-quantum cryptography. These give algorithms that we can run today on classical computers that defend against hypothetical quantum computers.

Certain very simple quantum algorithms (e.g., involving single qubits) we can run today in experimental settings. But more serious computations, such as quantum zero-knowledge proofs, we are very very very very very far from being able to run. Implementing such algorithms seem to require vastly larger quantum circuits than, e.g., Shor's algorithm.

The bad news

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. For example, factoring- and discrete-log-based systems typically require manipulating 2048- or 256-bit numbers, which is costly. Instead, we design symmetric-key cryptosystems based on ad-hoc assumptions. 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.

As so often in life, we want nice things. But nice things cost too much.

Operations with large numbers are costly primarily because the fastest multiplication algorithms for numbers of this size is *superlinear* in the number of bits in the number.

The schoolbook multiplication algorithm multiplies n-bit numbers in time $O(n^2)$. For RSA-sized numbers, you might use the fancier Karatsuba method in time $O(n^{1.58})$ but we don't typically go beyond that.

Why study the design of symmetric-key primitives?

- They are arguably the most important primitives in the practice of crypto.
  Used in: phone, computer, satellite, etc.
- The constructions are clever.
- They fit into our theme of post-quantum crypto.
- **NOT** so that you can write your own implementations or design your own ciphers.

NIST publishes standards for block ciphers. There are three widely used ones:

	Key size	Block size
DES (1975)	56 bits	64 bits
3DES	168 bits	64 bits
AES (1998)	128, 192, or 256 bits	128 bits

Some attacks exploit small block sizes. A 128-bit block size is the minimum and many would argue that it should be larger. Salsa20 uses a 256-bit key and a 512-bit block size.

Definition of PRF

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 $\mathcal{Y}$. For concreteness, we can think of $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$.

Intuitively, we think of a pseudorandom function as "looking 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 $f : \mathcal{X} \to \mathcal{Y}$.

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 $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, 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 $i = 1, 2, \ldots$ (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:

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

To turn this into a formal asymptotic definition, we parameterize the key space (also possibly the input/output spaces) by a security parameter λ, and we allow the adversary $\mathcal{A}$ to run in time $\mathrm{poly}(\lambda)$.

Definition 2 (Pseudorandom function)**.** A function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a pseudorandom function if all p.p.t algorithms $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that large enough $\lambda \in \mathbb{N}$,

$$\mathsf{PRFAdv}[\mathcal{A}, F_\lambda] \leq \mu(\lambda).$$

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

PRF vs. PRP A "block cipher" is really a pseudorandom *permutation* (PRP), rather than a pseudorandom *function*. The difference are:

- a PRP has the same input and output space: $P : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$,
- a PRP maps distinct inputs to distinct outputs—there are no collisions, and
- a PRP also has an efficient inversion algorithm given the key: $P^{-1}(k, \cdot)$.

AES and DES are block ciphers (a.k.a. PRPs). We use them in many cases as PRFs, since if an attacker sees T input/output pairs, it can only distinguish a PRP from a PRF with advantage roughly $T^2 / |\mathcal{X}|$.

If the output space of the PRP is large (e.g., 256 bits), then it might as well be a PRP for practical purposes.

Since nowadays, we typically construct things using PRFs rather than PRPs, I will not describe how the inversion algorithms for either construction works.

The general strategy

The plan Most of the symmetric-key primitives we use (e.g., block ciphers, hash functions) we design today using this process:

- **Step 1:** Assume access to an ideal primitive. (e.g., public random permutation on n bits)
- **Step 2:** Use the ideal primitive to build new primitive.
- **Step 3:** Instantiate the ideal primitive with something that we hope "behaves enough like" the ideal one.

The only "leap of faith" is in the third step. Without that, we typically can reduce security of the new primitive to that of the ideal one. The problem is that we need to eventually implement the ideal primitive with something, which requires a leap of faith.

Said another way, the PRF security of AES is effectively based on the assumption "AES is a secure PRF."

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 isn't security, but getting good performance on all sorts of different hardware. Many cryptographers would bet their lives that there will never be a 2^{40}-time attack on 256-bit AES. I don't think you will be able to find a cryptographer who will bet their life on the hardness of factoring or elliptic-curve discrete log.

Lattice assumptions are another matter. I think many people would bet at least their left hand that the learning-with-errors problem (which we will see soon) is hard.

DES

With explosion of potential commercial applications of cryptography, the predecessor to NIST published the "Data Encryption Standard" (DES) in 1975. The government never approved it for use in classified applications, as the 56-bit key length was too short even on the day the standard was published.

Reference note: This discussion is just a restatement of the descriptions of DES and AES in the Boneh-Shoup textbook. Consult their book for details.

Diffie and Hellman at the time pointed out that 56-bit keys were unacceptably short. Their analysis was based on projections about cheap computation would get and how quickly. They (presciently) proposed that using 128-bit keys would be prudent.

As far as I know, the best known attack on DES today is Matsui's linear cryptanalysis (1993), which recovers the key from 2^{47} input-output pairs in a known-plaintext attack.

To describe the design of DES using the plan we outlined in the previous section, we have to answer two questions:

- What ideal primitive will we use?
- How do we use it to build a PRP?

Step 1: The ideal primitive

The ideal primitive that the DES design uses is a set of 16 random functions $f_1, f_2, \ldots, f_{16} \colon \{0,1\}^n \to \{0,1\}^n$.

Step 2: Use ideal primitive to build PRF – Feistel network

DES then uses a very slick design proposed by Horst Feistel (MIT grad, at IBM at the time) now called the "Feistel network" (Fig. 1) to build a PRP out of a PRF.

The Feistel network builds a PRP out of a random function by applying a simple transformation many times (over many "rounds"). The Feistel permutation is $\pi \colon \{0,1\}^{2n} \to \{0,1\}^{2n}$ and, when instantiated with a function $f \colon \{0,1\}^n \to \{0,1\}^n$ is defined as:

$$\pi_f(x, y) := (y, x \oplus f(y)).$$

The neat property of the Feistel network is that it is invertible: it turned a random function into an (efficiently invertible) random permutation.

The initial and final bit shuffling in DES have no apparent effect on cipher's security. Boneh-Shoup speculates that the initial and final permutations were to slow down DES implementations in software relative to hardware. This is consistent with the widespread (and somewhat confirmed) view that DES was designed to be breakable by governments but good enough for commercial use.

Step 3: Instantiate the ideal primitive – final DES construction

DES uses a 56-bit keyspace ($|\mathcal{K}| = 2^{56}$) and a 64-bit block size ($2n = 64$).

The DES cipher on input $x \in \{0,1\}^{2n}$ just looks like:

$$\mathrm{DES}(x) := (P_1 \circ \pi_{f_{16}} \circ \cdots \circ \pi_{f_2} \circ \pi_{f_1} \circ P_0)(x),$$

where the f_is are the idealized random functions and P_0 and P_1 are just fixed permutations of the input bits.

Luby and Rackoff showed that if we take n off to infinity and model the f_is as random functions, even a three-round variant of DES is indistinguishable from a random permutation. Of course, their result says little about DES in practice, since (1) the f_is are not at all random and (2) we are dealing with very small input/output sizes and asymptotic results are not meaningful in this parameter regime.

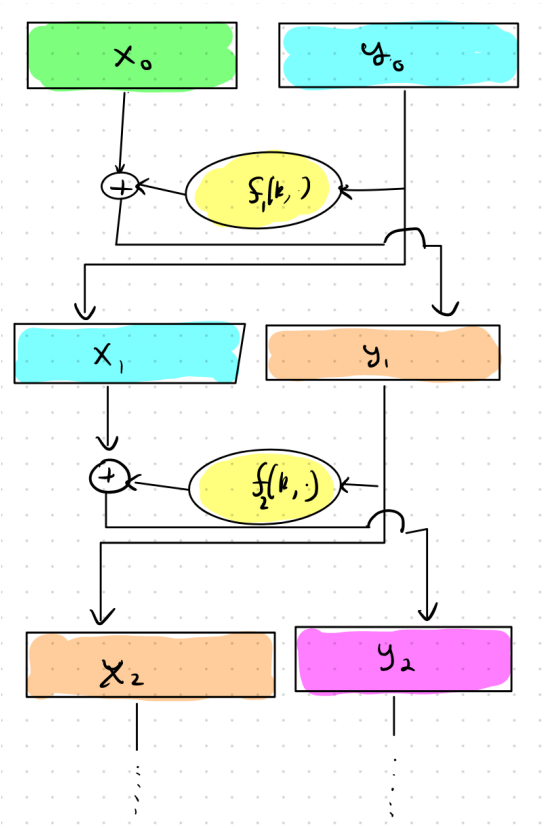

Figure 1: Feistel network

The round functions The last step is to instantiate the round functions (the f_is) with concrete functions.

To do so, DES first generates 16 "round keys" $k_1, \ldots, k_{16} \in \{0,1\}^{48}$, each derived from a subset of the bits of the 56-bit DES key k.

The round function $F(k_i, \cdot)$:

- computes the initial state as a linear function of the round key and its input,
- splits the state into six-bit chunks and applies a different *non-linear* function (called an S-box for "substitution") to each using a lookup table, and
- permutes the bits of the state.

The only non-linear part of the cipher is the S-boxes. It turns out that if you pick the S-boxes at random, the cipher becomes very weak. The DES S-boxes are constructed to have a bunch of nice statistical properties that prevent attacks. (For example, no output bit is not close to a linear function of the input bits.)

Instantiating the ideal DES construction with these 16 keyed round functions gives the final construction.

Lessons?

- Many rounds of a simple operation.
- Avoiding all of the known attacks takes a lot of care. Choosing parameters at random does not work.

AES

The AES block cipher uses a very different overall structure than DES, but the round function shares some similarities. AES is an "iterated Even-Mansour cipher." (AES with one round gives a cipher called the Even-Mansour cipher.) AES operates on a key of size $\{128, 192, 256\}$. The block size is a fixed 128 bits.

Unlike DES, the government uses AES to protect classified data. They use 128+ bits for SECRET data, and 192+ bits for TOP SECRET data.

Foreign governments who do not trust AES and design their own ciphers apparently often use weak homebrewed ciphers.

Step 1: The ideal primitive

AES uses an invertible permutation $\pi\colon \{0,1\}^n \to \{0,1\}^n$, which we can model as a truly random permutation.

Step 2: Use ideal primitive to build PRF

For this discussion, let the block size be n. AES first derives a number of "round keys" $k_0, \ldots, k_r \in \{0,1\}^n$ from the input key k. Each round key is a linear function of the input key k.

After that, the AES cipher on input $x \in \{0,1\}^n$ just alternates between XORing a round key into the state and applying the permutation π:

- $\mathsf{st} \leftarrow x \oplus k_0$
- For $i = 1, \ldots, r$:
  - $\mathsf{st} \leftarrow \pi(\mathsf{st}) \oplus k_i$.[1]
- Output st.

[1] The one detail we omit is that true AES uses a slightly different permutation π in the last round. This apparently enables some performance optimizations in hardware.

It is possible to show in an asymptotic sense, as the block and key sizes go off to infinity, this idealized AES construction behaves like a PRF. The Even-Mansour paper is a great read if you are interested in this.

Step 3: Instantiate ideal primitive – final AES design

Now to get to the full AES construction, we just need to instantiate the public permutation $\pi\colon \{0,1\}^n \to \{0,1\}^n$. It modifies the input in three steps:

- `SubBytes`. Use a lookup table $S\colon \{0,1\}^8 \to \{0,1\}^8$ hardcoded into the design to replace each of the 16 input bytes with a different one:

  $$b_1 \| b_2 \| \cdots \| b_{16} \quad \longmapsto \quad S(b_1) \| S(b_2) \| \cdots \| S(b_{16}).$$

  This is a non-linear operation. As in DES, the designers chose the S-box carefully to avoid various attacks.

- `ShiftRows`. View the 16-byte block as a 4×4 matrix. Perform a cyclic shift on this matrix: shift the 0th row 0 cells to the right, the first row 1 cell to the right, the second row 2 cells to the right, and the third row 3 cells to the right.

- `MixColumns`. View the 16-byte block as a 4×4 matrix. Multiply it by a fixed matrix.

The matrix multiplication in the `MixColumns` step is in $\mathrm{GF}(2^8)$. If you don't know what that means, it doesn't matter.

Cache attacks

Not only do you have to be careful in the *design* of ciphers like AES. You also have to be careful in the *implementation*. Software implementations of AES are particularly tricky.

The issue is in implementing the S-boxes, or other table lookups, in software. *Cache-timing attacks* are one serious pitfall. To explain: because of caching, if the AES routine makes consecutive lookups to $S[1]$ then $S[1]$, these will complete faster than if it looks up $S[1]$ then $S[187]$. The difference in timing—even though it is small—leaks

information about the internal state of the cipher. This can be enough to perform devastating attacks.

When using 128-bit keys, the number of rounds is $r = 10$.

Linear cryptanalysis

Note: Again, these notes are mostly a rephrasing of the content in Boneh-Shoup. Look there for the details.

The cryptanalysis of block ciphers is an art on its own. I will try to sketch the idea behind one sort of attack (based on the description of Matsui's attack as summarized in the Boneh-Shoup book), which may give you the flavor of how these attacks work.

Typically this cryptanalysis is performed against block ciphers. Since we're talking about PRFs today, I will stick with PRFs.

Let F be a PRF in which inputs, outputs, and keys are all n-bit strings. Further, let say that you find that there is some bias in the relationship between the key bits, input bits, and output bits.

A *linear relation* on the cipher E exists if there are sets of bit positions $B_x, B_y, B_k \subseteq [n]$ such that

$$\Pr\left[x[B_x] \oplus y[B_y] = k[B_k] \;:\; \begin{array}{c} x \xleftarrow{R} \{0,1\}^n \\ y \leftarrow F(k,x) \end{array}\right] \geq \frac{1}{2} + \epsilon,$$

where ϵ is noticeable.

In a truly random function, $F(k,x)$ is independent of x, so the bias $\epsilon = 0$. In a concrete PRF, the bias can be non-zero.

The idea of the attack is to gather a very large number of input-output pairs: $(x_1, y_1), \ldots, (x_T, y_T)$.

Then if we look at all of the input/output XORs, the linear relation tells us that the resulting values will be slightly biased towards the value of the B_kth bit of the key:

$$(x_1[B_x] \oplus y_1[B_y]), \ldots, (x_T[B_x] \oplus y_T[B_y]).$$

The idea is then to take the majority of these T values and use that value as our guess at the XOR of a subset of the key bits $k[B_k]$.

The Chernoff bound tells us that our guess will be right with probability at least $1 - \exp(T\epsilon^2/2)$. So if we have a linear relation with bias ϵ, we get a bit of information about the key with probability well over $1/2$ after seeing something like $T = 4/\epsilon^2$ input/output pairs.

In the case of DES, one linear relation depends on 12 bits of the secret key. If you have additional linear relations, you can use these to recover the full key. See Boneh-Shoup 4.3.1 for details.

References