Last Revised: 8 Jan 99.

ECS 227 — Modern Cryptography — Winter 99
Lecture 1

Introduction and Block Ciphers

7 January 1999

1 Introduction

Before we begin in earnest, perhaps it is nice to try to describe, however vaguely, what cryptography is all about. One nice description, dating back to 1976, is that:

> Cryptography is the study of "mathematical" systems for solving two kinds of security problems: privacy and authentication. [DiHe76]

That's not bad, though perhaps cryptography nowadays might be understood to include goals even beyond privacy and authentication. But certainly privacy and authentication are at the heart of things.

One thing which the above description omits is the central role of the *adversary*. Rivest says simply:

> Cryptography is about *communication in the presence of adversaries.* [Ri90]

Indeed, there is no cryptography in the absence of an adversary, and any rigorous treatment of cryptography elevates her to being a central character. One way to be sure that you've left cryptography is to realize that there isn't any adversary in sight.

Perhaps "adversary" is too negatively connoted. Us cryptographers usually like our adversaries. Without an adversary, at least a hypothetical one, we'd have nothing to do. We'd have to seek employment as mathematicians.

A crucial element of cryptography is *exploiting asymmetry.* The asymmetry to which we refer is between what the "good guys" possess or can do and what the adversary possesses or can do. Most typically, the asymmetry is realized by *keys* which good guys have but the adversary lacks. But the asymmetry can be realized in other ways. For example, it could be a channel which is more reliable for the good guy than it is for the adversary. Or it could be computing resources which are greater for the good guys than the adversary. Cryptography is about finding ways to exploit such asymmetries to do useful things.

1.1 Classical Cryptography

Today's lecture introduces two "branches" of cryptography: I'll call them *classical cryptography* and *reductionist cryptography.* This course will concentrate on the latter. I want to touch on classical cryptography today as a way to put reductionist cryptography in proper perspective.

Classical cryptography is an attack-centric approach which works something like this:

(1) A cryptographic problem is recognized.

(2) Then a solution is offered.

(3) Then one searches for an attack on the proposed solution.

(4) When one is found, if it is deemed damaging or indicative of a potential weakness, go back to Step 2 and try to come up with a better solution.

The third step is called *cryptanalysis*.

The above process continues until one regards the solution as "adequate." In commercial contexts one must of course implement the solution and ship the products which embody the cryptographic solution. With luck, damaging attacks will not subsequently materialize. Sometimes they do, and when this happens the company which owns the product may find it difficult to effectively fix the fielded solution. See Figure 1.

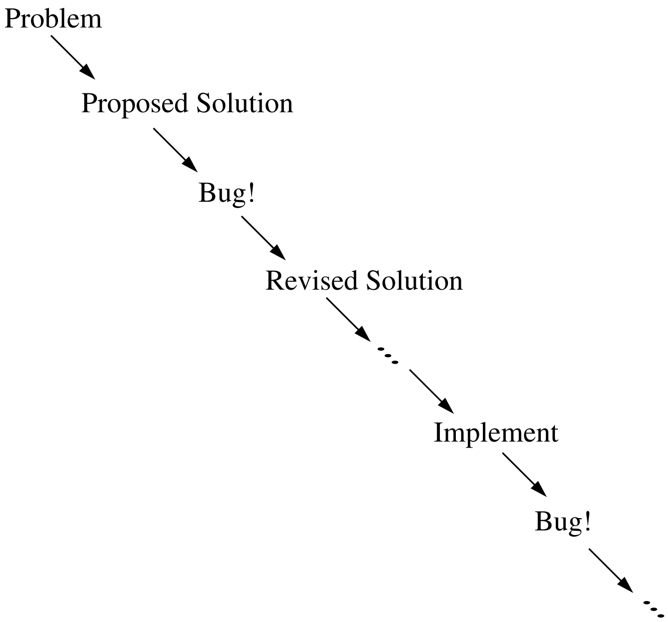

Figure 1: The classical-cryptography approach.

Doing classical cryptography well takes great cleverness. Sometimes one finds problems with cryptographic primitives in the form of subtle mathematical relationships which allow one to subvert the protocol's aims. Sometimes, instead, one "jumps out of the system," showing that some essential cryptographic issue was overlooked in the design, application, or implementation of the cryptography.

1.2 Links to Complexity Theory

The classical approach is rather disappointingly *ad hoc.* Is this characteristic unavoidable in the field? One might be tempted to believe so, perhaps because of the following piece of fallacious reasoning:

> A correct solution to typical cryptographic goals implies the existence of a one-way function (or similar, concrete-complexity results). The existence of a one-way function, in turn, implies $\mathbf{P} \neq \mathbf{NP}$. We can not expect firm foundations for a field whose most

basic of goals already imply complexity-theoretic results stronger than anyone has any idea how to show.

The error in the above reasoning is manifest in the fact that complexity theory *itself* rests on firm foundations (and has made central contributions to computer science) *despite* the unresolved nature of its most basic questions. One reason has been the power of *reductions*—most visibly, the theory of NP-completeness.

When we show that some problem Π is NP-Complete, we say something interesting about Π *even though we have not managed to show* $\mathbf{P} \neq \mathbf{NP}$. In reductionist cryptography, when we show that a proposed solution is "secure," we do the exact same thing. The analogy is spelled out in Figure 2.

We think that computational problem Π can't be solved in polynomial time.	We think that cryptographic scheme Π can't be effectively attacked.
We believe this because if Π could be solved in polynomial time, then so could problem SAT (say).	We believe this because if Π could be effectively attacked, then so could DES (say).
To show the above, we *reduce* SAT to Π: we show that *if* we could solve Π in polynomial time, then we could also solve SAT in polynomial time.	To show the above, we *reduce* DES to Π: we show that *if* we could break Π by effective means, then we could break DES by effective means.

Figure 2: The analogy between reductionist-cryptography and NP-Completeness.

The particulars of reductions in cryptography are harder to comprehend than they were in elementary complexity theory. Part of the difficulty for us will be that every problem domain will have it's own unique notions of an "effective attack"— rather like having a different "version" for the notion of NP-Completeness as you move from one problem to the next. We will also be concerned with the *quality* of reductions. One could have concerned oneself with this in complexity theory, but it's not usually done. For doing practical work in cryptography, however, paying attention to the quality of reductions is really very important.

Let us now expand on reductionist security just a bit more.

1.3 Reductionist Security

Reductionist security (also called *provable security*) once again begins with (1) recognizing some real-world cryptographic problem. But next (2) we formulate a *definition* for the problem at hand. Next we provide (3) a protocol, and then we (4) prove, (3.5) under a complexity theoretic assumption, that the protocol meets its definition (2). Now, if we want we (5) implement. See Figure 3.

The definition step (2) can be the most difficult step (and often the most under-valued.) In some cases, it is sufficiently difficult as to require a further abstraction boundary for us feeble-minded humans. In that case we split the process into two. First (1.5) we describe a *model* for the problem; and only then do we do a definition. The model is "where" the problem is defined.

The above discussion is all pretty abstract, but examples will be given throughout this class.

1.4 A Layered Look at Secure Systems

A reductionist approach to cryptography has to have something from which to reduce. For the purposes of this class we imagine a layered architecture for secure systems. See Figure 1.4

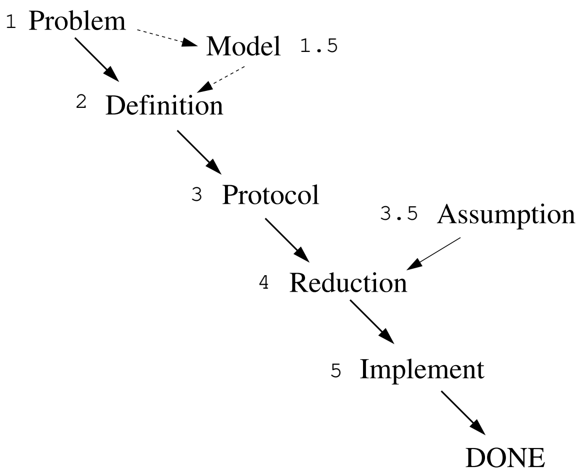

Figure 3: The provable-security approach.

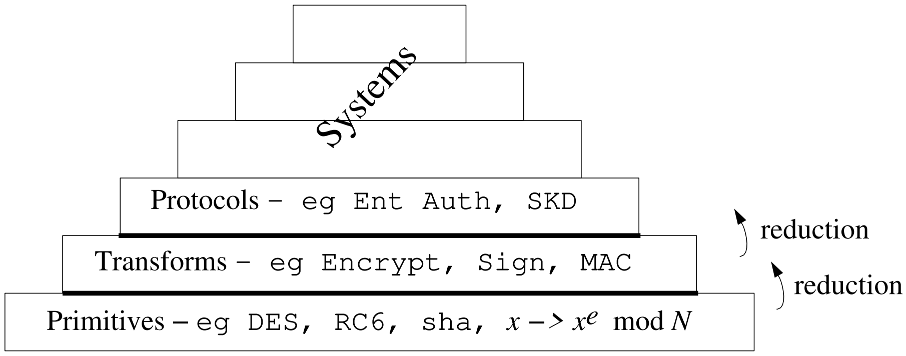

Figure 4: Architectural layers in a secure system

At the bottommost layer are *cryptographic primitives*. An example of a cryptographic primitive is the algorithm of the Data Encryption Standard (DES), which maps a 56-bit string together with a 64-bit string to another 64-bit string. A second example is the candidate AES submission RC6. A third example is the compression function of SHA-1, which maps a 160-bit string together with a 512-bit string to another 160-bit string. All of these functions have "weird" definitions—a few pages of description, but a very strange few pages they are. They are examples of *confusion/diffusion primitives*. A different sort of example of a cryptographic primitive is the "RSA-function," which is the map $x \mapsto x^e \bmod N$, where N is the product of two odd primes and $e \in \mathsf{Z}_{\phi(N)}$. This is an *algebraic* primitive, not a confusion/diffusion one.

Cryptographic primitives are the "lowest level" mappings which cryptographic practice provides. It is important to keep clear that they don't, by themselves, do anything of practical interest. For example, even if our goal is to encrypt, DES is not, by itself, an algorithmic solution to this problem.

Cryptographic primitives are not amenable to reductionist security: there is nothing to reduce to them.

The next layer up is what we might call *cryptographic transforms*. These are the algorithms which do things of cryptographic interest. The most important examples are transforms to encrypt messages and authenticate messages. Other important transforms are pseudorandom generators and one-way functions.

Cryptographic transforms can be treated in the reductionist approach, and that is exactly what we shall do. We will show things in the spirit: if RC6 is secure then some-particular-way-to-encrypt-with-RC6 is secure.

At the next layer up are *cryptographic protocols.* These are multi-round exchanges of messages aimed at some more complex cryptographic goal. Common examples are entity authentication (how two parties come to decide they are speaking to each other), key distribution, and voting schemes. Once again, the reductionist approach can be used to treat security. Here one would typically reduce cryptographic transforms to the desired protocol goal. Occasionally one may want to directly reduce a cryptographic primitive to the protocol goal.

Finally, there are *systems.* These comprise both hardware and software and typically employ as cryptographic underpinnings a variety of cryptographic protocols. Conceivably the reductionist approach could be applicable to security systems, but this has never been demonstrated. We will only be looking at reductions from primitives to transforms, and from transforms to protocols. Keep in mind that most real-world security problems occur up here, in the systems territory which we will not explore.

1.5 A Menagerie of Cryptographic Problems

Now that the approach has been vaguely sketched, let's look at the sort of *problems* cryptographers like to think about. I won't pay any attention to possible protocols for these problems, nor will I care about the problems' definitions. Be clear that what's given below are simple *descriptions* of cryptographic problems (step 1, above), not definitions (step 2). A definition must be completely unambiguous, partitioning the world into solutions and non-solutions, or quantifying a candidate solution.

1.5.1 Message Privacy

The days in which cryptography meant *encryption*—transformations to preserve *privacy*—are long over, and yet encryption is the classical problem of cryptography, so we follow tradition and begin there. In this problem Alice (A) wants to send a secret message M to Bob (B) over a wire on which an adversary Eve (E) is listening. Bob should understand the message; Eve shouldn't.

You might hope for encryption to emulate the properties of a perfect lead pipe. Alice can whisper a message on her end, Bob can hear the message on his end, and Eve sees nothing but lead. This idealization is too much to hope for. Encryption is not intended to hide the *existence* of a message—a potentially important piece of information to the adversary. Nor is encryption intended to hide the *length* of a message—another potentially interesting piece of information to the adversary. If we knew a maximal length for message we could use padding to try to hide the length of messages, but this would typically entail a large loss of efficiency. We're not going to do that. *Traffic analysis* refers to the adversarial behavior of gaining information by looking at who is sending messages of what length to whom. Encryption doesn't protect you from traffic analysis.

To make the encryption goal achievable there will have to be some *asymmetry* between Eve and Bob. For if Eve and Bob had identical initial information and capabilities, and receive identical communications from Alice, then everything Bob could figure out Eve could figure out, too.

Information asymmetry can be achieved in a number of ways. In *symmetric* (private-key) encryption, Alice and Bob share a key a which Eve lacks. Possession/non-possession of the key is the information asymmetry which symmetric encryption exploits. Alice sends Bob a message by computing $C \leftarrow \mathcal{E}_a(M)$. The *encryption function* $\mathcal{E}$ may be probabilistic or stateful. *Probabilistic* means that the outcome of computing the function depends on random coin tosses. *Stateful* means that the outcome of computing the function depends on the function values which have already been computed. Bob recovers M by *decrypting* the incoming message, setting $M' \leftarrow \mathcal{D}_a(C)$. Of course one expects that, when $C \leftarrow \mathcal{E}_a(M)$ and then $M' \leftarrow \mathcal{D}_a(C)$, one has that $M = M'$.

A picture for symmetric encryption can be found in Figure 5. I warn that it is in some sense already misleading. In particular, the arrow going from the communications channel to Eve would seem to suggest that Eve is just a passive eavesdropper. In fact, in our formalization, Eve will not be passive.

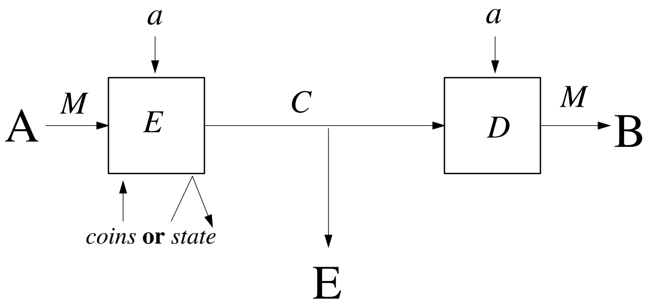

Figure 5: Symmetric encryption.

There are other ways to set up the information asymmetry between Bob and Eve. In *asymmetric encryption* Bob possesses a *pair* of keys —a *public key*, PK_B, and a *secret key*, SK_B. Bob's public key is made publicly known and bound to his identity. When Alice wants to send a secret message M to Bob, she computes $y \leftarrow \mathcal{E}_{\mathrm{PK}_B}(M)$. Again, $\mathcal{E}$ may be probabilistic or stateful. When Bob receives a message y, he computes $\mathcal{D}_{\mathrm{SK}_B}(y)$. Of course one expects that, when $y \leftarrow \mathcal{E}_{\mathrm{PK}_B}(M)$ and then $M' \leftarrow \mathcal{D}_{\mathrm{SK}_B}(y)$, one has that $M' = M$.

The *trust model* specifies who, initially, has what keys. We have just described two different trust models for achieving the same basic aim.

1.5.2 Message Authenticity

In this problem Bob receives some message which is claimed to have originated with Alice. The channel on which this message flows is insecure. Thus Bob wants to distinguish the case in which the message really did originate with Alice from the case in which the message originated with some forger F

Once again, information asymmetry is needed. The symmetric and asymmetric trust models described above are equally applicable here.

The usual tool for solving the message authentication problem in the symmetric setting is a *message authentication code* (MAC). Here Alice and Bob share a secret key, a, and when Alice wants to send a message M to Bob she attaches to it a short string σ (this is called a *tag* or *MAC*.) This is computed using M and a. The MAC computation might be probabilistic or stateful, as with encryption, or it may well be deterministic. Bob, on receipt of M and σ, uses a to check if the tag is OK. If so, he accepts M as authentic; otherwise, he regards M as a forgery. See Figure 6. The picture is again slightly misleading, but it gets the point across.

The tool for solving the message authentication problem in the asymmetric setting is a *digital signature.* Here Alice has a public key PK_A and a corresponding secret key SK_A. Everyone (even the adversary) knows the key PK_A and that it belongs to Alice. When Alice wants to send a message M to Bob she attaches to it a short string σ (called a *signature*) computed as a function of M and SK_A. The signing computation might be probabilistic or stateful, or it may well be

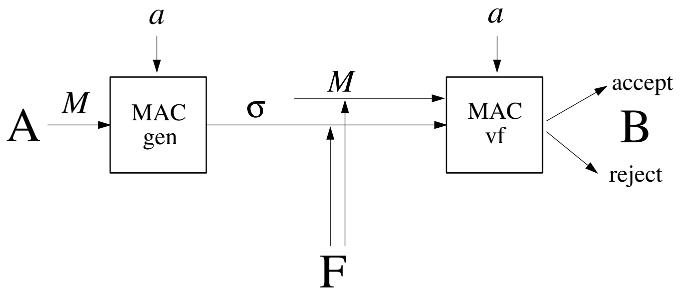

Figure 6: Message authentication code.

deterministic. Bob, on receipt of M and σ, checks if it is OK using the public key PK_A. If it is fine, he accepts M as authentic; otherwise, he regards M as a forgery.

1.5.3 Pseudorandom Generator (PRG)

Computers are quite deterministic. But for lots of applications, "random numbers" are quite useful. These applications involve simulation, efficient algorithms, and cryptography itself.

How can a completely deterministic machine generate "random" numbers? It can't. But a machine can do the next best thing: it can *stretch* a little bit of randomness into a lot of "pseudo-randomness."

Suppose we wire to our computer a Geiger counter that generates a "random" bit every second. We run our computer for a little while and now it has 200 "random" bits. We won't worry about the "philosophical" question as to whether these bits are random in any *real* sense. We'll simply assume that these bits are completely unpredictable to anything "beyond" the computer which has gathered this data—mathematically, we'll treat these bits as random.

A *pseudorandom generator* (PRG) stretches a short (eg., 100-bit) "truly random" string into a much longer (a million bits, say) string which "looks" random.

Is this problem cryptography? If so, where's the adversary?!

The adversary shows up when you define the goal. We will be quite ambitious and say that the stretched string should look random to any reasonable algorithm—any computationally feasible method whose function in life is to make an intelligent guess: "I think this is a million random bits" OR "I think this is a pseudorandom generator applied to a random 100-bit string." This *distinguisher* is the adversary.

This problem has a simple enough definition that we might as well give it. An *adversary* D is an algorithm. Let $g : \{0,1\}^\ell \to \{0,1\}^L$ be a function (our PRG) where $\ell < L$. Consider providing adversary D one of two things: $g(r)$, for a random string $r \in \{0,1\}^\ell$; or else R, where R is a random string of length L. It is D's job to try to figure out which of those two things we just did. Adversary D should answer "1" if it thinks that we've given it a pseudorandom input (ie., a $g(r)$ value for a random r) while it should answer "0" if it thinks we've given it a truly random input (ie., an R-value for a random R). We define the advantage of D as

$$\mathrm{Adv}_D^{\mathrm{prg}} = \Pr[r \leftarrow \{0,1\}^\ell : D(g(r))] - \Pr[R \leftarrow \{0,1\}^L : D(R)]$$

This is a measure of how well the adversary is doing. Zero means that the adversary is doing terribly; she could get this advantage by always answering "0" or by flipping a coin. An advantage of one would mean that the adversary is doing great; she always gets it right. (For this problem that is impossible—what is the best that an adversary can actually do?).

For a PRG to be good we would like for $\mathrm{Adv}_D^{\mathrm{prg}}$ to be small no matter what the adversary D might do, as long as D runs in some reasonable amount of time. This is a generalization of all those statistical tests described in Knuth volume 2, and in other places. We can define $\mathrm{Adv}_g^{\mathrm{prg}}(t)$ to be the maximal value of $\mathrm{Adv}_D^{\mathrm{prg}}$ over all adversaries that run in $\leq t$ computational steps. (Forgive the overloaded notation.) This is a measure of the quality of the PRG g. We would like that $\mathrm{Adv}_g^{\mathrm{prg}}(t)$ to be small (eg, $\leq 2^{-20}$) until t becomes enormous (eg, $t \geq 2^{60}$). We won't be able to prove such a thing for any concrete generator—but it won't really matter.

Congratulations; you have now seen your first definition!

1.5.4 Telephone Coin Flip

Alice and Bob want to decide which of them has to show up to AI-class tomorrow. They take turns going to class, you see, but there are an odd number of lectures. Neither wants to go, but one of them better do it.

Alice calls Bob on the telephone and offers a simple solution. "Bob," she says, "I've got a penny in my pocket. I'm going to toss it in the air right now. You call *heads* or *tails*. If you get it right, I'll go to AI class and take notes for us. If you get it wrong, you'll have to go."

Bob's not as bright as Alice, but something troubles him about this arrangement ...

The *telephone coin flip* problem is to come up with a *communications protocol* so that, to the maximal extent possible, neither Alice nor Bob can cheat the other; and, at the same time, each of them learn the outcome of a fair coin toss.

Here is a solution—sort of. Alice puts a random bit α inside an envelope and sends it to Bob. Bob announces a random bit β. Now Alice opens the envelope for Bob to see. The shared bit is defined as $\alpha \oplus \beta$. See Figure 7

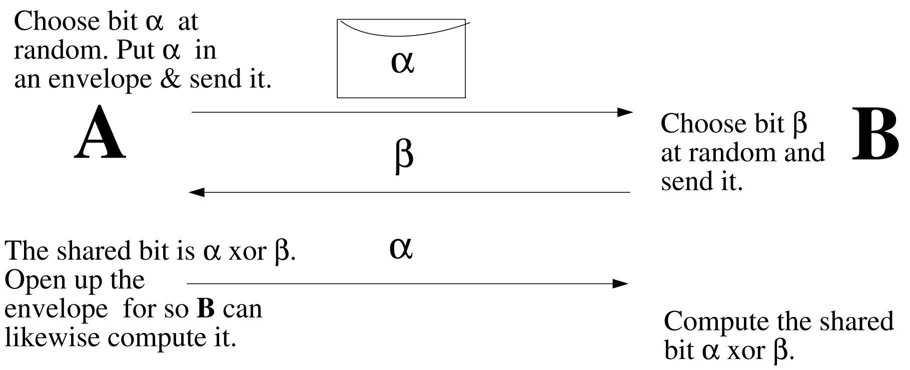

Figure 7: Envelope solution to the telephone coin flip problem.

To do this over the telephone we need some sort of "electronic envelope" (in cryptography, this called a *commitment scheme*). Alice can put a value in the envelope and Bob can't see what the envelope contains. Later, Alice can open the envelope so that Bob can see what the envelope contains. Alice can't change her mind about an envelope's contents—it can only be opened up in one way.

Here is a simple technique to implement an electronic envelope. To put a "0" inside an envelope Alice chooses two random 100-bit primes p and q subject to the constraints that $p < q$ and, $p \equiv 1 \pmod 4$ and $q \equiv 3 \pmod 4$. The product of these, $N = pq$, is the commitment to zero; that is what Alice would send. To put a "1" inside an envelope Alice chooses too random 100-bit primes p and q subject to the constraints that $p < q$ and $p \equiv 3 \pmod 4$ and $q \equiv 1 \pmod 4$. The product of these, $N = pq$, is the commitment to one. Poor Bob, seeing N, would like to figure out if the smaller of its two prime factors is congruent to 1 or to 3 modulo 4. We have no idea how to make that determination short of factoring N—and we don't know how to factor 200 digit numbers which are the product of random 100-digit primes—our best algorithms would still take way too long. When Alice wants to decommit (open the envelop) N she announces p and q. Bob verifies that they are prime (this is easy to do) and multiply to N, and then he looks to see if the smaller factor is congruent to 1 or to 3 modulo 4.

1.5.5 Dating Problem

Alice might—or might not—want to go out on a date with Bob. Bob might—or might not—what to go out on a date with Alice. The age-old problem (which keeps many of us apart) is a desire not to have A's (resp., B's) interest in B (resp., A) known to B (resp., A) unless B (resp., A) is likewise interested. That is, Bob would rather not even ask Alice if she's going to say no—and Alice would rather not even ask Bob if he's going to say no. But if no one ever asks, how is anyone to ever know? Solving the dating problem is finding a way to compute the AND of privately held bits in such a manner that only the correctly computed conjunct is known to the two parties.

I won't give a solution to the dating problem—I don't know a simple solution in the communications model. See if you can find one!

2 Block Ciphers

I'd like to describe one of the most well-known kinds of primitives—a block cipher. Block ciphers are used for symmetric encryption, MACs, and for many other goals as well. To give you some sense of what block ciphers look like, I'll sketch two concrete ones: DES and RC6.

2.1 Two Breeds of Primitives

Concrete primitives of contemporary cryptography come in two basic flavors:

1. **Confusion/Diffusion Primitives.** DES and RC6 (and things like SHA, too) are examples. I've heard these primitives called *bit-twiddling constructions.* That may sound a bit flippant, but, in fact, it aptly describes the flavor of their definitions. Typically one uses straight-line code (if the rounds are unrolled) of Boolean or computer-arithmetic operations, trying to create computational hardness by doing enough "strange stuff."

2. **Algebraic/Combinatorial Primitives.** RSA and knapsack-based ciphers are two examples. Here one starts with a "known" computational problem and uses it as the basis for cryptography. Mathematical domains which have been used to provide these computational problems include number theory, computational group theory, automata theory, and the theory of error correcting codes. Number theory seems to get most of the attention.

Some theoretical computer scientists have little regard for the confusion/diffusion approach. They view it as somehow "inelegant" when compared to primitives like RSA. Personally, I think the bias against confusion/diffusion primitives is outdated and unfair. For one thing, the "mathematical structure" which accounts for the elegance of many algebraic/combinatorial primitives provides a potential route for attack. Second, confusion/diffusion primitives tend to be more efficient. This, together with their apparent arbitrariness, gives confusion/diffusion primitives a type of elegance all their own.

Some cryptographic aims aren't known to be solvable starting from a confusion/diffusion primitive. In particular, we don't know how to achieve asymmetric encryption under a confusion/diffusion primitive, nor the possibly simpler problem of *secret key exchange* under a confusion/diffusion primitive.[1] In other domains, including digital signatures, it is known how to achieve a solution using a confusion/diffusion primitive, but the solutions are complex and less efficient than solutions which make use of algebraic primitives in addition to a confusion/diffusion primitive.

The distinction we have drawn between confusion/diffusion primitives and algebraic/combinatorial ones is not a distinction we know how to formalize. Indeed an algebraic/combinatorial primitive can always be realized by a (large) Boolean circuit which is doing the same sort of bit-twiddling operations of a typical confusion/diffusion primitive. Despite this, when you are presented with a new primitive you'll immediately recognize if its of the confusion/diffusion flavor, or of the algebraic/combinatorial sort.

2.2 Block Cipher Syntax

A block cipher is a function

$$F : \{0,1\}^\alpha \times \{0,1\}^\beta \to \{0,1\}^\beta$$

where $F(a, \cdot)$ is a permutation for any $a \in \{0,1\}^\alpha$. The number α is called the *key length* and the number β is called the *block length*. Usually we write $F_a(x)$ in lieu of $F(a, x)$.

2.3 The Data Encryption Standard (DES)

The algorithm of the U.S. Data Encryption Standard was developed at IBM around 1974. It became a U.S. NBS Standard in 1977. The algorithm of this standard is sometimes called the Data Encryption Algorithm (DEA), and sometimes it is called again by DES. The algorithm maps a 64-bit *plaintext* to a 64-bit *ciphertext* under control of a 56-bit *key*. We'll denote this map by

$$\mathrm{DES} : \{0,1\}^{56} \times \{0,1\}^{64} \to \{0,1\}^{64}.$$

Usually we write the first argument (the key) as a subscript, preferring $\mathrm{DES}_a(x)$ to $\mathrm{DES}(a, x)$.

2.3.1 Definition of DES

The complete specification of DES is in FIPS Pub 46 and in a handout I distributed. In class we talked through the definition. We described, in particular, how any construction of the same basic structure —a *Feistel* cipher— yields a *permutation* on $\{0,1\}^\beta$ for each key. (That is, the particulars of the function F is irrelevant for achieving this.) We also described why DES^{-1} could be computed by simply reordering the subkeys $K_{16}, \ldots, K_1$.

We certainly won't repeat DES's definition in these notes. Do look at the handout to get a feel for this. Coppersmith has an article which may demystify some of DES's structure; see me for a copy.

2.3.2 Attacks on DES

There has been much work aimed at breaking DES. To date, the most effective "mathematical" attacks are from differential cryptanalysis (see the Biham and Shamir book) and linear cryptanalysis (see Mitsuro Matsui, Eurocrypt '93 and CRYPTO '94). Neither of these techniques is practical,

[1] In this problem A and B communicate over a channel subject to eavesdropping (only) by adversary E. Parties A and B want to come into possession of a shared, random, secret key.

requiring an enormous number of (plaintext, ciphertext) pairs. (Differential cryptanalysis of DES requires $\approx 2^{47}$ pairs, while linear cryptanalysis uses about $\approx 2^{43}$ pairs.) Currently, the most practical known attack is exhaustive search: the key space is just 2^{56}, which can be exhausted in a large, but not overwhelming, amount of computational work. In fact, the Electronic Frontier Foundation (EFF) has recently built a DES-cracking machine as a demonstration project.

The ideas underlying differential and linear cryptanalysis are interesting but they would take us a bit too far afield of our main efforts. Perhaps later we shall schedule a lecture on these topics.

2.4 Definition of RC6

See the handout given in class by Rivest *et al.*—especially the picture.

2.5 Comparing DES and RC6

DES was designed using classified information by individuals at IBM and the NSA; RC6 was designed using unclassified information by individuals who will describe whatever they know. DES was designed to be hardware-efficient, while RC6 was designed to be software-efficient. DES uses a Feistel network; RC6 uses a modified Feistel network. DES has modest-length keys; RC6 supports very long keys. DES has a block size of 64 bits; RC6 has a block size of 128 bits. In DES, "confusion" is primarily accomplished with the eight S-boxes, while "diffusion" is primarily accomplished with the wire-crossing P; in RC6, confusion and diffusion are accomplished with data-dependent rotations, computer addition and computer multiplication.

2.6 Block Cipher Security

We did not talk about what a block cipher is supposed to do—but we shall see!

3 Symmetric Encryption

In the final few minutes of class we started to look at how a block cipher might be used to encrypt. I will include this discussion with the next set of notes.

References

Overviews.

[DiHe76] Whitfield Diffie and Martin Hellman, "New Directions in Cryptography." *IEEE Transactions on Information Theory,* vol IT-22, No. 6, November 1976.

[Go97] Oded Goldreich, "The Foundations of Cryptography—An Essay." Full version of invited paper appearing at CRYPTO '98. `http://theory.lcs.mit.edu/~oded/foc.html`

[Ri90] Ronald Rivest, "Cryptography." From the *Handbook of Theoretical Computer Science,* edited by J. van Leeuwen, Elsevier Science Publishers B.V., 1990.

Block ciphers. I described two examples: the old classic, DES, and a nice new construction, RC6.

[DES] FIPS 46, "Data Encryption Standard." Federal Information Processing Standards Publication 46, U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Springfield, Virginia, 1981.

[**RC6**] "The RC6 Block Cipher," Ronald Rivest, Matt Robshaw, Ray Sidney, and Yiquin Lisa Yin. Unpublished submission submitted for consideration as the new AES.

Some of the design criteria for DES are described in:

[**Co92**] "The Data Encryption Standard (DES) and its Strength Against Attacks," Don Coppersmith, IBM Research Report RC 18613, December 1992.