Week 1

Substitution Ciphers and Information Theory

Definition 1.1 (cryptography, cryptology). Techniques involving the generation, storage, and transfer of information providing properties such as:

- privacy (confidentiality)
- authentication (data or person)
- non-repudiation (eg. signatures that can't be denied)

Rivest: "Cryptography" is communication in the presence of adversaries."

History

Cryptography is more than 4000 years old. This century highlighted by new applications to and demands from cryptography due to computers and new communication methods. See Kahn — "The Codebreakers" and Singh — "The Code Book."

- 1920's — Friedman applied mathematical techniques to code-breaking (cryptanalysis)
- 1940's — Shannon introduced rigorous mathematical concepts (information theory)
- 1970's
  - DES proposed as a standard (IBM)
  - Public-key cryptography introduced by Diffie and Hellman (including digital signatures)
  - RSA scheme proposed
- 1980's and on — lots of new scientific methodology and new applications
- 2001 — AES accepted as standardized successor to DES

Example 1.1. Zero-knowledge proof systems. Suppose two entities (A and B) know a particular n-node graph, and A knows a Hamiltonian cycle on the graph. B knows the graph is Hamiltonian, but doesn't know a path.

A zero-knowledge protocol allows A to prove to B that he knows a Hamiltonian cycle without revealing the actual cycle to B.

Can be used as the basis for authentication / access control schemes.

1.1 Substitution Ciphers

We will begin with some simple schemes and see why they don't/do work.

Method 1

Message of characters (ASCII 7-bit encoding). Choose a random offset $0 \leq K \leq 127$ (for example).

Encrypt: shift characters by K (add K mod 128). Eg. $K = 2$. HELLO encrypts to JGNNQ

Decrypt: subtract K

The value of K must be kept secret.

Note. $K = 3$ — Caesar cipher

This scheme might work for very few characters (say 1), but is very insecure for more characters. For example, suppose $E_K(\text{THE RAIN IN SPAIN}) = C = \text{XYWZP\%S T3R}\ldots$. Compute $D_0(C), D_1(C), \ldots, D_{128}(C)$ (try all possible keys). For sufficiently long plaintexts, chances are good that only a single value of K will yield meaningful English. (Prob of n character message being English is about $1/2^{3.2n}$.)

Definition 1.2 (Unicity distance). The minimal message length necessary to ensure only one decryption yields meaningful English text

For English (using the above cipher), this is about 25-28 characters (different for other languages and other ciphers).

For short plaintexts, there may be many possible decryptions (eg. HAL, IBM, etc...).

The same kind of problem happens with *any* small key space (our example has only 7 bits).

Method 2

Instead of *fixed* offsets, use a random permutation of $\{0, \ldots, 127\}$ for the offsets.

How many permutations (keys)? 128! ($\log(128!) \approx 716$ bits of key)

Trying 10^{18} per second results in only 10^{25} attempts per year. Therefore, an exhaustive search attack is not possible (would require about 10^{190} years).

Nevertheless, this cipher can still be broken using other language statistics. For example, in English, character frequencies tend to be

$$\text{E} — 12.51\%, \text{T} — 9.25\%, \text{A} — 8.04\ \%, \ldots, \text{Z} — 0.09\%$$

Try setting the most frequent ciphertext symbol to E, then T, etc... With a little effort, such a cipher can be broken easily (even if the ciphertext doesn't fit frequencies exactly). For example, the book "A Void" has no letter e! Other statistics, like digraph and trigraph frequencies (TH vs HT) can also help.

One moral: A large key space is necessary, but not sufficient.

Method 3

Try random substitutions of *pairs* of ciphertexts. For example:

$$\text{JA CK AN DJ IL LX} \mapsto \text{RZ PW}\ldots$$

Still breakable by higher-order statistics, but harder (longer keys — $\log(128^2)! \approx 205748$, larger unicity distance — 64297)

Trigraph (triplets), quartets, etc... are even harder to break, but still breakable.

1.2 Entropy

Definition 1.3 (Shannon). The *entropy* of a probability distribution ($\sum p_i = 1$, $0 \le p_i \le 1$) is

$$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i \quad (0 \log 0 = 0) \ .$$

Intuition: measures the amount of *uncertainty* in an event, or the amount of information that's really contained when it occurs.

Example 1.2.
$H(0, 1, 0, 0) = 0$ — sure of outcome (event 2)

$H(0.01, 0.98, 0.01, 0) = \epsilon$ — event 2 is most likely (almost certain)

$H(1/4, 1/4, 1/4, 1/4) = 2$ (bits) — maximum uncertainty (all 4 events are equally likely)

$H(1/2, 1/4, 1/8, 1/8) = 1.75$ — in-between case

$H(1/2^n, 1/2^n, \ldots, 1/2^n) = n$

Theorem 1.1. *A stream of n symbols, independently distributed with entropy $H(p)$ can be compressed to an $H(p)n$-bit string.*

Compressing n characters of English: $7n$ ASCII bits to $1.5n$ compressed bits.

Rate (bits of information per character):

- English — 1.5 bits
- Random (compressed) text — $\log 26 \approx 4.7$ bits (randomly distributed)

Redundancy is the difference between the absolute rate (bits of information per character of random text) and the actual rate. For English, the redundancy is about $4.7 - 1.5 = 3.2$.

Relative redundancy: redundancy/absolute rate = 1 − rate/abs rate For english, about 0.68.

1.3 Unicity Distance

Definition 1.4. The *unicity distance* of a cipher is the amount of ciphertext needed until the encryption key is unique. Depends on the key entropy and redundancy of the plaintext.

Let's suppose that the language is English (entropy per character of 1.5 bits). Thus, for n characters, the entropy is $1.5n$. There are approximately $2^{1.5n}$ n-character English strings, which are a subset of the $26^n \approx 2^{4.7n}$ random n-character strings. Therefore, if n characters are chosen at random, the probability that valid English is obtained is about

$$\frac{2^{1.5n}}{2^{4.7n}} = \frac{1}{2^{3.2n}} \to 0 \text{ as } n \to \infty \ .$$

Now, given $C = E_{K_0}(M)$ for M n-character English, assume that

$$\forall K \neq K_0, \ D_K(C) \text{ is a random } n\text{-character string.}$$

The probability of being English by chance is $1/2^{3.2n}$.

Assume that the key is m bits long. Then the expected number of coincidental English decryptions is

$$2^m \left(\frac{1}{2^{3.2n}} \right) = 2^{m-3.2n}$$

which is < 1 if $m - 3.2n < 0 \Longrightarrow n \geq m/3.2$.

For key length 56, this is about 18.

For key length 128, this is about 40.

Example 1.3. What is the unicity distance for English encoded in ASCII (say 7 bits per character). Assume random permutation (128! keys) and 3.2 bits per 7-bit character.

$716/(7 - 3.2) \approx 189$

Complexity of Exhaustive Search

To turn the preceding into a code-breaking algorithm:

Input: ciphertext $C = E_K(M)$

Construct a boolean circuit with input the m key bits. The circuit applies D to the ciphertext and passes it through an English-tester (easy to construct) which outputs 1 if the text is English, 0 otherwise.

To find the key, we need to find a satisfying assignment — easily accomplished in $O(2^m)$ steps (exhaustive search - intractable for large m).

This is the SAT problem — if $P = NP$, then any such scheme is breakable in polynomial time.

1.4 Cryptanalysis

Question: What does it mean to "break" a system?

1. Recover key?
2. Recover whole plaintext?
3. Recover *some* of plaintext?
4. Recover parity of all plaintext bits (but no individual bits)?
5. Do the above sometimes/always?

Example 1.4. What if data is random binary strings (rather than English characters)? Eg. Swiss banks assign 100-bit *random* strings as account passwords. Anybody with the correct string can access the account (the chance of guessing someone's password is pretty small). Suppose the password is enciphered with a substitution cipher.

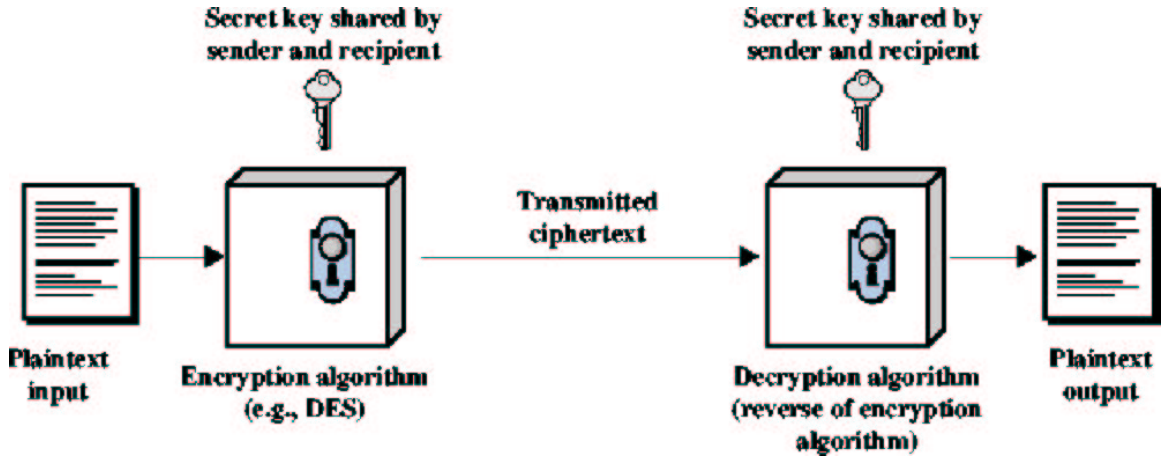
- Pick a random permutation on $\{0, 1\}$ (1-bit key):
  - Π_0 — identity,
  - Π_1 — $1 \mapsto 0$, $0 \mapsto 1$
- Apply permutation to each character (i.e., bit). For example, plaintext 010011011011 enciphers to:
  - $k = 0$ — 010011011011
  - $k = 1$ — 101100100100

Can you estimate the key from the ciphertext? No!

Can you guess the first bit of plaintext? No!

Can you guess the parity of plaintext bits? Yes!

1.4.1 Schematic of a conventional (one-key or symmetric) cryptosystem



Encryption function E_K

Decryption function D_K

Clearly, we must have $D_K(E_K(M)) = M$ for all M

Assume Eve (the eavesdropper) knows what E, D are, but *not* K, which is randomly chosen. (What does it mean to *not* know? $H(K) = n$ large where $K \in \{0,1\}^n$.)

Justification for this assumption:

- playing it safe
- no loss of generality — if E, D are secret, consider this knowledge as part of the key

	extra key bits
(E_1, D_1)	00
(E_2, D_2)	01
(E_3, D_3)	10
(E_4, D_4)	11

Also, Eve may have *prior information* about plaintext, such as:

- English, Japanese, etc...
- Plaintext begins with "Sept 9, 1999," "Dear Charlie," "Please sell," etc...

Types of Cryptanalytic Attack

1. Ciphertext Only Attack (COA) — the cryptanalyst possesses only the ciphertext
2. Known Text Attack (KTA) — the cryptanalyst has *some* corresponding plaintext and ciphertext. For example:
   - Diplomatic proposal: it is known that such proposals are usually sent back by the foreign diplomat to her capital word for word in the original language. If you know which ciphertext corresponds to which diplomatic proposal, you have knowledge of some corresponding plaintext and ciphertext.

- Timed press release: big corporations often wish a press release to be issued simultaneously worldwide, so they transmit it to their offices in encrypted form, together with information on when to decode it. If you have a copy of this ciphertext, you can (perhaps) match it up with the press release.

3. Chosen Text Attack (CTA) — the same as KTA, but the cryptanalyst is given plaintext and corresponding ciphertext *of his own choosing.* For example:
   - Diplomatic proposal: if you take an unexpected action, you can be reasonably certain that it will be reported by the foreign diplomat, and you can probably manipulate the form of the report.
4. Adaptive Chosen Text Attack — attacker is able to select new plaintext/ciphertext pairs based on previous decryptions

Goals of Eve

Determine

- key
- plaintext
- partial information about the plaintext

Week 2

Perfect Security and Block Ciphers

2.1 Perfect Security

Suppose an eavesdropper Eve has the following information:

possible plaintexts	Eve's prior probability	Eve's prob after seeing ciphertext C
M_1	$p(M_1) = p_1$	$q_1 = p(M_1 \mid C)$
M_2	p_2	q_2
M_3	p_3	q_3
$\vdots$	$\vdots$	$\vdots$
M_n	p_n	q_n

Definition 2.1. A cryptosystem is *perfectly secure* if $p(M) = p(M \mid C)$, or $H(P) = H(Q)$.

Intuition: the ciphertext C gives an attacker no information about the message that was sent.

Equivalent definition: $p(C) = p(C \mid M)$.

Note.

$$p(C) = \sum_{k \in \mathcal{K}} p(K) p(D_K(C))$$

$$p(C \mid M) = \sum_{\substack{k \in K \\ E_K(M) = C}} p(K)$$

Recall our Swiss bank example (assuming ciphertext only attack):

Password	Prior prob	Prof after ciphertext
$000\ldots0$	$1/2^{100}$	0
$000\ldots1$	$1/2^{100}$	0
		$1/2$
$\vdots$	$\vdots$	$\vdots$
		$1/2$
$111\ldots1$	$1/2^{100}$	0
	$H(P) = 100$	$H(Q) = 1$

With ABCD string: 24 possible keys, 200 character strings

Prior	After
p_1	q_1
p_2	q_2
$\vdots$	$\vdots$
$p_{4^{200}}$	$q_{4^{200}}$
$H(P) = 800$	$H(Q) \leq \log 24 = 4.58$

Only 24 of the q_i are non-zero.

Fact: if $(q_1, q_2, \ldots, q_n) = (0, 0, r_1, r_2, 0, \ldots, r_3, 0)$ then $H(Q) = H(r_1, r_2, r_3)$.

2.1.1 The One-time Pad

Let M be any n-bit message and K be any n-bit (random) key. Let C be the ciphertext given by $C = M \oplus K$. This cipher is called the *one-time pad*, and is provably secure as long as the following hold:

1. K must be random
2. K must be as long as M
3. K must be used only once

Suppose K were used twice:

$$\begin{aligned} C_1 &= M_1 \oplus K \\ C_2 &= M_2 \oplus K \\ \implies C_1 \oplus C_2 &= M_1 \oplus M_2 \quad \text{since } K \oplus K = (0, 0, \ldots, 0) \ . \end{aligned}$$

Note that $C_1 \oplus C_2 = M_1 \oplus M_2$ is nothing more than a coherent running key cipher (adding two coherent texts, M_1 and M_2), which is insecure.

Why is this perfectly secure? For all plaintext messages M and ciphertexts C, $\text{Prob}(M|C) = \text{Prob}(M)$ (proof for homework).

Moscow-Washington hotline uses this, but it is very expensive and difficult to implement.

Theorem 2.1 (Shannon 1949). *With an n_1-bit uniformly random key K, and plaintext entropy n_2,*

1. *If $n_2 \leq n_1$, the perfect security is possible,*
2. *If $n_2 > n_1$, then perfect security is impossible.*

Proof (Sketch).

1. Use one-time pad, compressing plaintext to n_2 bits if necessary.
2. If there are n possible messages:

prior	after
p_1	q_1
p_2	q_2
$\vdots$	$\vdots$
p_n	q_n
$H(P) = n_2$	$H(Q) \leq n_1 < n_2$

   (since at most 2^{n_1} of the q_i are non-zero).

□

2.2 Block Ciphers

Types of ciphers:

1. Stream ciphers — one bit enciphered at a time, dependent on history
2. Block ciphers — blocks of n bits enciphered simultaneously. Designed to be independent of history, implemented to be dependent on history

A *block cipher* is a cipher of the form $E_K(M) = C$, $D_K(C) = M$. $\{0,1\}^m \times \{0,1\}^n \mapsto \{0,1\}^n$ (key/plaintext to ciphertext).

- block length $n = 64, 128, 192, 256$
- key length m "reasonably" large (in practice $128, 192, 256$)
- Reuse keys for *many* blocks (say, billions)

How good can such ciphers be? We already know they're terrible if $n = 7$.

We can always detect if a block repeats. Serious? Maybe (eg. chosen plaintext — $2^{128} \approx 2 \times 10^{38}$ plaintexts — too many!)

This problem can be avoided by *preventing* repetitions. Eg. number each message block (32-bit texts — 32-bit integer number). Then 100 billion bits before we run out of distinct numbers.

Better method: cipher-block chaining (cover later)

Consider the "ideal" case: E_K is a random permutation on $\{0,1\}^n$ (indexed by K). The number of possible permutations (keys) is

$$2^n! \approx \sqrt{2\pi 2^n} \left(\frac{2^n}{e}\right)^{2^n}$$

(from Stirling's approximation) and the key entropy is

$$\log(2^n!) \approx 2^n n \ .$$

Therefore, key entropy is about 10^{40} when $n = 128$

Good news: unicity distance is really large ($10^{40}/3.2$)

Bad news (obvious): with such long keys, might as well use one-time pad.

Goal: get by with a much shorter key (key length asymptotically less than message length)

This immediately implies that the unicity distance will be small, so such a cipher is breakable in principle with small ciphertext.

Caveat: the actual algorithm is in NP, so it may still be computationally hard to break the cipher.

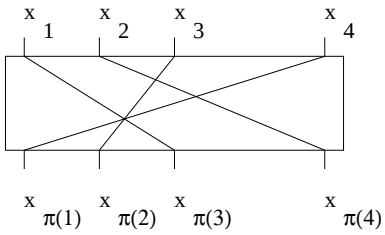
Note. Breakable in polynomial time if $P = NP$. Thus, proving computational hardness (i.e., not breakable in polynomial time) also proves $P \neq NP$. May be tough to do!

Two simple ideas for designing block ciphers (that don't work):

1. Random re-ordering of n bits (transposition)
2. Random substitution

2.2.1 Transposition

$(x_1, x_2, \ldots, x_n) \mapsto (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$



Π is a permutation on $\{1, 2, \ldots, n\}$, not $\{0,1\}^n$.

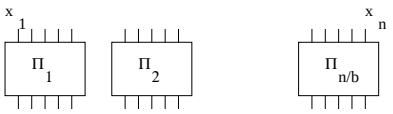
Thus, there are $n!$ permutations, so the key entropy is about $n \log n$.

Good news: shorter keys (296 bits when $n = 64$)

Bad news: easy to break with known plaintext attack (assignment 2)

2.2.2 Random Substitutions

Do random substitutions on small blocks of text.



Say 8-bit blocks (bytes) — at most $\log(2^8!) \approx 1684$ key bits per block and at most 13472 key bits in total (assuming 64-bit blocks).

Good news: keys a bit long but not terrible.

Bad news: easily breakable

2.2.3 Product Ciphers

Alternate transposition and substitution for a few rounds (say 16).

Transposition (diffusion) — broken by linear structure

Substitution (confusion) — nonlinear, but broken by "locality"

This methodology is more heuristic than rigorous, but it defeats our obvious attacks. If details are carefully chosen, can defeat more powerful attacks such as differential and linear cryptanalysis, even with fairly small keys.

2.3 Data Encryption Standard (DES)

$n = 64$ (block size), $m = 56$ (key length actually 64 bits, 8 of which are parity bits)

1. The 64 plaintext bits are permuted in a fixed order (transposition cipher).

2. The block is divided into two 32-bit words L_0 and R_0.
3. The block undergoes 16 substitution "rounds." In each round, one word is transformed using XOR and a substitution function, after which the two words are swapped.
4. In the last round, the two words are not swapped.
5. The original permutation is reversed.

Thus

$$DES_{key}(M) = IP^{-1}(S_{16}(S_{15}(\dots(S_2(S_1(IP(M))))\dots))) \ .$$

Initial Permutation IP

See FIPS publication. Notation: first bit of the output is the 58th bit of the input.

Substitution Rounds

In round i, the right word R_{i-1} is combined with the ith subkey K_i via the function f. The output of f is XORed with L_{i-1} to form R_i. The next left word L_i is the previous right word ($L_i = R_{i-1}$).

The Function f

f accepts as input R_i (32 bits) and the ith round subkey K_i (48 bits). The subkey generation is described below. The function f works as follows:

1. R_i is expanded to the 48 bit $R_i^{'}$ via the expansion function E, which simply repeats some of the bits of R_i in generating $R_i^{'}$ (see the FIPS publication for the specification of E).
2. $R_i^{'}$ is XORed with K_i (both are 48 bits long).
3. $R_i^{'} \oplus K_i$ is broken into 8 6-bit words. Each of these words is replaced by a 4-bit word according to the 8 (different) S-boxes $S_1, S_2, \dots, S_8$. The result of applying the S-boxes is a 32-bit string.
4. The 32-bit string is permuted according to the fixed permutation P (see the FIPS publication).

Generation of the Subkeys K_i

1. The key K (56 bits) is permuted according to the fixed permutation "PERMUTED CHOICE 1" (see FIPS publication) and separated into two 28-bit words C_0 and D_0.
2. Each word is rotated either one or two places to the left according to the fixed schedule below, yielding C_1 and D_1.

Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# of left shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

3. K_1 is obtained from C_1 and D_1 via "PERMUTED CHOICE 2," which selects 48 bits from C_1 and D_1 according to a fixed ordering.
4. Steps 2 and 3 are repeated with C_i and D_i to obtain the remaining 15 subkeys.

Figure 2.1: Diagram of DES

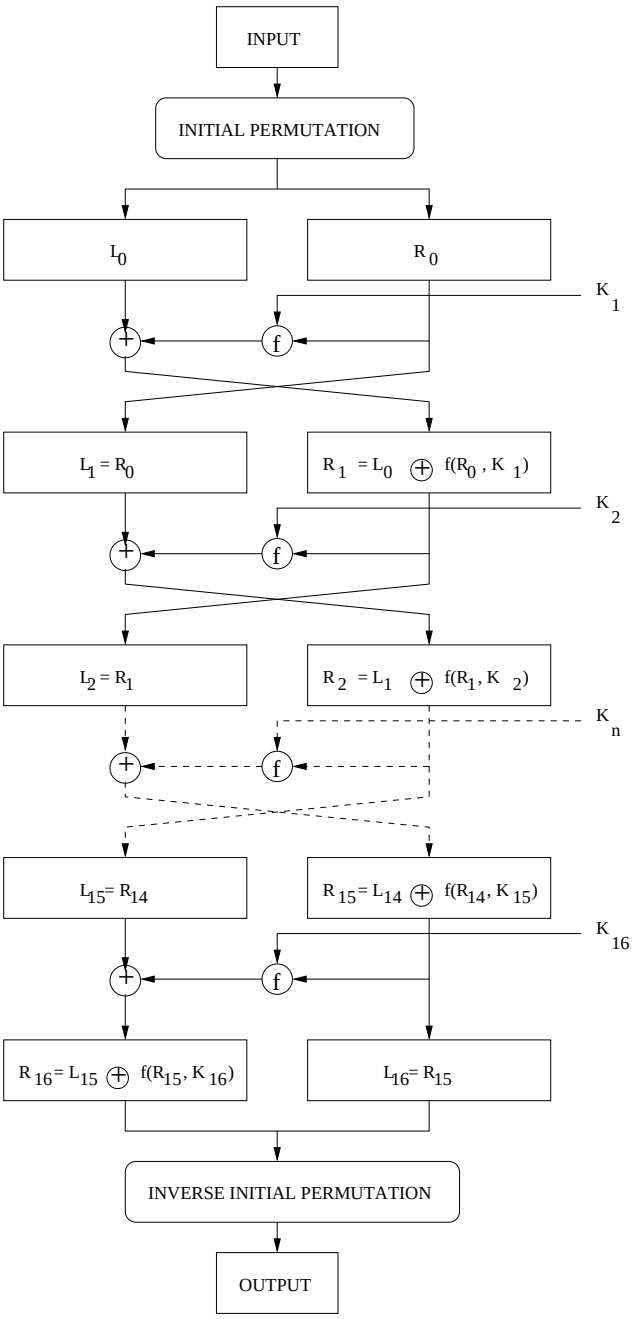


Figure 2.2: DES function f

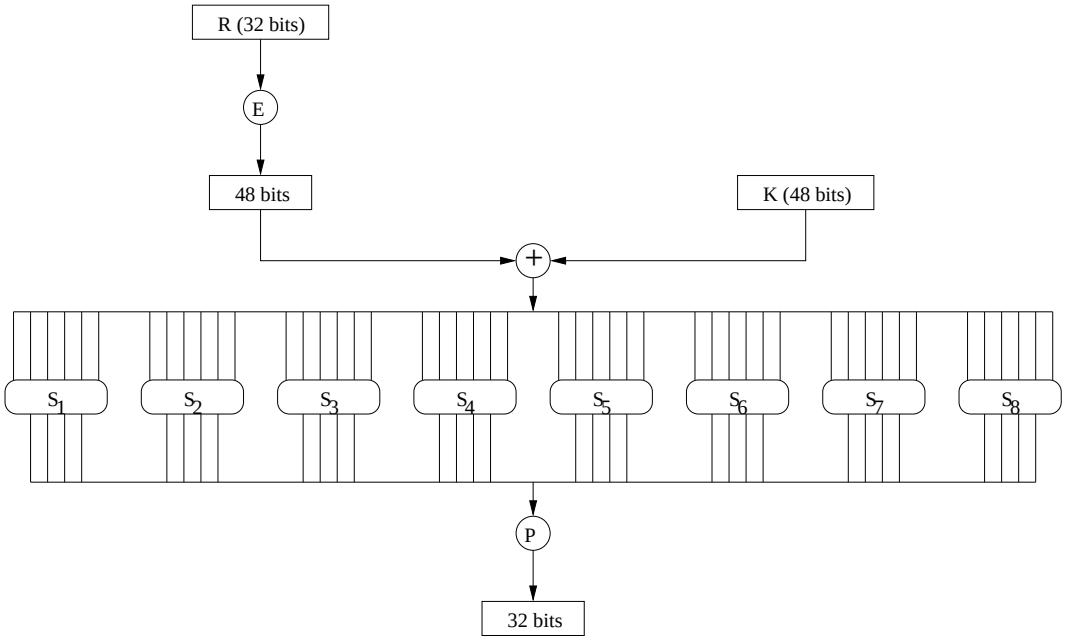
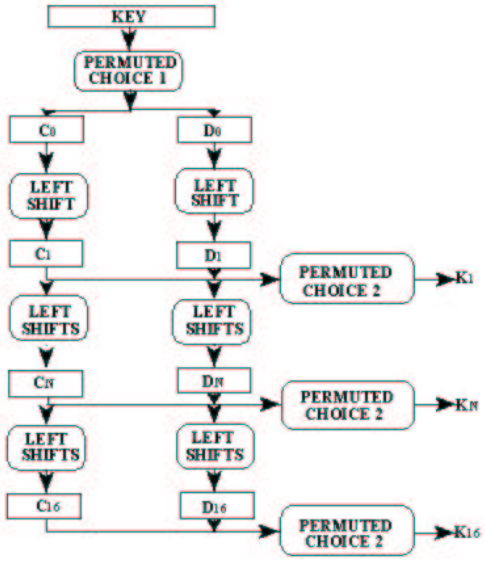


Figure 2.3: DES Key Schedule Algorithm



Formal Notation and Decryption

$K_n = KS(n, key)$ (K_n is the nth subkey, KS is the key schedule)

Note:

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus f(R_i, K_{i+1}) \quad i = 0, 1, \ldots, 15 \\ L_i &= R_{i+1} \oplus f(R_i, K_{i+1}) \\ R_{i+1} \oplus L_i &= f(R_i, K_{i+1}) \end{aligned}$$

$$\begin{aligned} C &= IP^{-1}(R_{16}, L_{16}) \\ IP(C) &= (R_{16}, L_{16}) \end{aligned}$$

(IP^{-1} is the inverse of the initial permutation function)

Suppose $DES_{key}(M) = C$. Denote by

$$K_i^{'} = KS(17 - i, key), \quad i = 1, 2, \ldots, 16$$

(the key schedule in reverse order). Now run the DES device on C, using K_i' instead of K_i. We have $IP(C) = (R_{16}, L_{16})$, and thus

$$\begin{aligned} L_0^{'} &= R_{16} \\ L_1^{'} &= R_0^{'} = L_{16} \\ &= R_{15} \end{aligned}$$

$$\begin{aligned} R_0^{'} &= L_{16} \\ R_1^{'} &= L_0^{'} \oplus f(R_0^{'}, K_1^{'}) \\ &= R_{16} \oplus f(L_{16}, K_{16}) \\ &= R_{16} \oplus f(R_{15}, K_{16}) \\ &= R_{16} \oplus R_{16} \oplus L_{15} \\ &= L_{15} \end{aligned}$$

In fact, by continuing this argument we get

$$L_i^{'} = R_{16-i} \quad R_i^{'} = L_{16-i}$$

and hence

$$IP^{-1}(R_{16}^{'}, L_{16}^{'}) = IP^{-1}(L_0, R_0) = M$$

Decryption of DES is simply running the DES algorithm on C with the reverse key schedule.

Note. The invertibility of DES is *independent* of the function f. Regardless of what function is used for f, decryption of DES works exactly as described above. This works largely because the individual parts of DES are *involutions* — functions that are their own inverses ($g(g(x)) = x$)

Week 3

Triple DES and The Advanced Encryption Standard

3.1 Cryptanalysis of DES

Currently-known attacks:

- exhaustive key search — 2^{55} computations (ciphertext only)
- linear cryptanalysis — 2^{43} (known plaintexts)
- differential cryptanalysis — 2^{47} (chosen plaintexts)

Designers (IBM and NSA) knew of some of these attacks, but kept it secret.

Very fast hardware implementations of DES, DES chips. The following were proposed, but never built:

- Diffie 1977, 10^6 DES chips, \$20 million US, 12 hours
- Wiener 1993, 57000 DES chips, \$1 million US, 3.5 hours

Electronic Frontier Foundation has built a DES cracker for \$250000 which finds a single DES key in 56 hours (tests 8800 keys per μsec). A combination of the DES cracker and 100000 PC's on the internet has found a DES key in 22.25 hours (tests 245000 keys per μsec).

3.2 Modes of Operation

FIPS 81

1. Electronic Code Book (ECB) — encrypt plaintext block independently
2. Cipher Block Chaining (CBC) — plaintext blocks are XORed with previous ciphertexts before encrypting
$$C_i = E_K(M_i \oplus C_{i-1}), \quad M_i = D_K(C_i) \oplus C_{i-1}$$
Set $C_0 = IV$, a random 64-bit initialization vector (sent in encrypted form to the receiver).
Advantages:

- Repeated plaintext blocks are encrypted differently
- Can be used for message authentication (last ciphertext block depends on all plaintext blocks)

3. Cipher Feedback (CFB) — self-synchronous stream cipher, same advantages as CBC

$$C_i = M_i \oplus z_i, \quad z_i = E_K(C_{i-1})$$

$C_0 = IV$ is a 64-bit random initialization vector, can be transmitted in the clear. Can also work with k-bit (rather than 64-bit) feedback.

4. Output Feedback (OFB) — synchronous stream cipher

$$C_i = M_i \oplus z_i, \quad z_i = E_K(z_{i-1})$$

Errors in a given plaintext block do not affect subsequent blocks (good for unreliable communication channels).

3.3 Triple DES

Try two successive DES encryptions (112-bit key):

$$C = DES_{K_1}(DES_{K_2}(M))$$

Doubling key length *squares* exhaustive search time — 2^{112} currently intractable.

No extra security if $DES_{K_1} DES_{K_2} = DES_{K_3}$ (i.e., if DES is a group). Campbell and Wiener (1992) proved that DES is *not* a group.

Nevertheless, this still doesn't work due to man-in-the-middle attack. Suppose we have two known plaintext pairs (M_1, C_1) and (M_2, C_2).

1. Make an array of $DES_{K_i'}(M_1) = D_i$ for all possible DES keys K_i' with $A[D_i] = K_i'$.
2. Sort the table (or use a hash table).
3. Pick K_2 at random until $A[DES_{K_2}^{-1}(C)] = K_1$ is non-zero.
4. Confirm the values of K_1 and K_2 with (M_2, C_2).

Total number of steps: $O(2^{56})$ — not much more secure than single DES.

Still impractical, but double DES is certifiably weaker.

(FIPS sepcial publication 800-67) Try three successive DES operations (168-bit key):

$$C = DES_{K_1}(DES_{K_2}^{-1}(DES_{K_3}(M)))$$

Advantages:

- Same as single key if $K_2 = K_1$ or $K_2 = K_3$.
- Exhaustive search has complexity 2^{112} (but with a 168-bit key).
- No other known practical attacks.

Main problems: slow (especially in software), fixed block size (64 bits)

3.4 The Advanced Encryption Standard

$n = 128$, $m = 128, 192, 256$, world-wide royalty-free availability

Public (world-wide) process of candidate submission and evaluation

Candidates were selected according to:

- security — best attack should be exhaustive key search,
- cost — speed and memory efficiency in software, hardware, and smart cards,
- algorithm and implementation characteristics (simplicity and elegance).

21 algorithms were submitted on June 15, 1998, of which 15 were announced as candidates on August 20, 1998. Five finalists, MARS, RC6, Rijndael, Serpent, and Twofish, were selected in August 1999.

3.4.1 Rijndael

FIPS 197

Rijndael (developed by Daemen and Rijmen) was chosen as the AES.

- designed for block sizes and key lengths to be any multiple of 32, including those specified in the AES
- iterated cipher: number of rounds, N_r depends on the key length. $N_r = 10$ for $m = 128$, $N_r = 12$ for $m = 192$, and $N_r = 14$ for $m = 256$ (see p. 14 of NIST documet).
- $\mathbb{F}_{2^8} = \mathbb{F}_2/(x^8 + x^4 + x^3 + x + 1)$ used for non-linear byte operations.
- the algorithm operates on a 4×4 array of bytes called the *state*:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

  The dimensions of the state depend on the block size.

The Rijndael algorithm (given plaintext M) proceeds as follows (p. 9):

1. Initialize `State` with M :

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$\leftarrow$

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

   where M consists of the 16 bytes $m_0, m_1, \ldots, m_{15}$.
2. Perform AddRoundKey, which XOR's the first `RoundKey` with `State`.
3. For each of the first $N_r - 1$ rounds:
   - Perform SubBytes on `State` (using an S-box on each byte of `State`),
   - Perform ShiftRows (a permutation) on `State`,
   - Perform MixColumns (a linear transformation) on `State`,

- Perform AddRoundKey.

4. For the last round:
   - Perform SubBytes,
   - Perform ShiftRows,
   - Perform AddRoundKey.
5. Define the ciphertext C to be `State` (using the same byte ordering).

Note. Rijndael is a product cipher: each round contains subkey mixing (AddRoundKey), substitution (SubBytes), and a permutation (ShiftRows).

The SubBytes Operation

(p.15) Each byte of `State` is substituted (independently). Can be implemented via table lookup (memory permitting), but is described algebraically. Let ϕ be the function mapping bytes to elements of $\mathbb{F}_{2^8}$ defined by

$$\phi : (a_7 a_6 \dots a_0) \mapsto \sum_{i=0}^{7} a_i x^i, a_i \in \mathbb{F}_2 \ .$$

Then:

$$\text{SubBytes}(a) = \phi^{-1}\left[(x^4 + x^3 + x^2 + x + 1)\phi(a)^{-1} + (x^6 + x^5 + x + 1) \bmod (x^8 + 1)\right] \ .$$

This operation can be performed using the following steps:

1. $z = \phi(a)$ (field representation of the byte a)
2. $z = z^{-1}$ (take the inverse in $\mathbb{F}_{2^8}$)
3. $b = \phi^{-1}(z)$ (map the field element z to the byte b)
4. Output the byte b' using the following affine transformation:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Note that

$$b'_i = b_i \oplus b_{i+4 \bmod 8} \oplus b_{i+5 \bmod 8} \oplus b_{i+6 \bmod 8} \oplus b_{i+7 \bmod 8} \oplus c_i$$

where $c = (11000110)$.

The inverse of SubBytes (called InvSubBytes, p. 22) is defined by

$$\text{InvSubBytes}(a) = \phi^{-1}\left[((x^6 + x^3 + x)\phi(a) + (x^2 + 1) \bmod (x^8 + 1))^{-1}\right] \ .$$

The ShiftRows Operation

(p. 17) Shifts the rows of `State` by 0, 1, 2, or 3 cells to the left:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

The inverse operation InvShiftRows (p. 21) applies right shifts instead of left shifts.

The MixColumns Operation

(p. 17) Consider each column of `State` as a four-term polynomial with coefficients in $\mathbb{F}_{2^8}$. For example:

$$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \mapsto s_{3,0}y^3 + s_{2,0}y^2 + s_{1,0}y + s_{0,0} = col_0(y) \ .$$

Let $a(y) = (x+1)y^3 + y^2 + y + (x)$ be fixed. Then the MixColumns operation replaces each column of `State` via

$$col_i(y) \leftarrow a(y)col_i(y) \pmod{y^4+1}, \quad i = 0, 1, 2, 3 \ .$$

Note. MixColumns can also be described as a linear transformation applied to each column of `State`, i.e., multiplying each 4-element column vector by a 4×4 matrix with coefficients in $\mathbb{F}_{2^8}$.

The inverse (called InvMixColumns, p. 23) is given by

$$col_i(y) \leftarrow a(y)^{-1}col_i(y) \pmod{y^4+1}, \quad i = 0, 1, 2, 3$$

and can also be described as a linear transformation.

AddRoundKey and the Key Schedule

In AddRoundKey (p. 23), each column of `State` is XORed with one word of the round key:

$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline w_{0,i+0} & w_{0,i+1} & w_{0,i+2} & w_{0,i+3} \\ \hline w_{1,i+0} & w_{1,i+1} & w_{1,i+2} & w_{1,i+3} \\ \hline w_{2,i+0} & w_{2,i+1} & w_{2,i+2} & w_{2,i+3} \\ \hline w_{3,i+0} & w_{3,i+1} & w_{3,i+2} & w_{3,i+3} \\ \hline \end{array}$$

Here $w_{i+0} = (w_{0,i+0}, w_{1,i+0}, w_{2,i+0}, w_{3,i+0})$ is the first round key for round i, made up of four bytes.

AddRoundKey is clearly it's own inverse.

Consider 128-bit Rijndael. There are 10 rounds plus one preliminary application of AddRoundKey, so the key schedule must produce 11 round keys, each consisting of four 4-byte words, from the 128-bit key (16 bytes). KeyExpansion (p. 19) produces an expanded key consisting of the required 44 words. In the following, the key $K = (k_0, k_1, k_2, k_3)$, where the k_i are 4-byte words, and the expanded key is denoted by the word-vector $(w_0, w_1, w_2, \ldots, w_{44})$.

1. for $i \in \{0, 1, 2, 3\}$, $w_i = k_i$
2. for $i \in \{4, \ldots, 44\}$:

$$w_i = w_{i-4} \oplus \begin{cases} \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \text{Rcon}_{i/4} & \text{if } 4 \mid i \\ w_{i-1} & \text{otherwise} \end{cases}$$

The components of KeyExpansion are:

- RotWord is a one-byte circular left shift on a word.
- SubWord performs a byte substitution (using the S-box SubBytes on each byte of it's input word).
- Rcon is a table of round constants (Rcon_j is used in round j). Each is a word with the three rightmost bytes equal to 0.

KeyExpansion is similar for 192 and 256-bit keys.

Decryption

To decrypt, perform cipher in reverse order, using inverses of components and the reverse of the key schedule:

1. AddRoundKey with round key N_r
2. For rounds $N_r - 1$ to 1 :
   - InvShiftRows
   - InvSubBytes
   - AddRoundKey
   - InvMixColumns
3. For round 1 :
   - InvShiftRows
   - InvSubBytes
   - AddRoundKey using round key 1

Note. Straightforward inverse cipher has a different sequence of transformations in the rounds. It is possible to reorgainize this so that the sequence is the same as that of encryption (see A2).

Strengths of Rijndael

Secure against all known attacks (best known is exhaustive key search).

Non-linearity resides in S-boxes (SubBytes):

- linear approximation and difference tables are close to uniform (thwarting linear and differential cryptanalysis)
- no fixed points ($S(a) = a$) or opposite fixed points ($S(a) = \overline{a}$)
- not an involution ($S(S(a) \neq a$)

ShiftRows and MixColumns ensure that after a few rounds, all output bits depend on all input bits.

Secure key schedule:

- knowledge of part of the cipher key or round key does not enable calculation of many other round key bits
- each key bit affects many round key bits

Very low memory requirements

Very fast (hardware and software)

Weaknesses

Decryption is slower than encryption (not a problem for CFB, OFB, or MAC generation).

Decryption algorithm is different from encryption (requires separate circuits and/or tables).

3.5 Other Block Ciphers

IDEA, SAFER K-128, RC5, Blowfish (all with 128-bit keys or more)

- Seem resistant to known attacks
- Exhaustive key search intractable
- Fast hardware and/or software implementations

Block ciphers are an important building block of modern security applications. We've learned:

- roughly how existing block ciphers work (product cipher)
- outline of kinds of attacks
- potential pitfalls
- heuristic nature of security

We haven't learned:

- enough to design our own block ciphers (haven't considered attacks in enough depth)

Week 4

Number Theory and Algorithms

4.1 Linear Diophantine Equations

Solve $ax + by = 1$ given $a, b \in \mathbb{Z}, b > 0$, and $\gcd(a, b) = 1$. If $\gcd(a, b) \neq 1$, the equation is insoluble. If $b < 0$, use $-b$ and solve for $(x, -y)$.

Euclidean Algorithm

$$\begin{aligned}
a &= bq_0 + r_0 && q_0 = \lfloor a/b \rfloor, 0 < r_0 < b \\
b &= r_0 q_1 + r_1 && q_1 = \lfloor b/r_0 \rfloor, 0 < r_1 < r_0 \\
r_0 &= r_1 q_2 + r_2 && q_2 = \lfloor r_0/r_1 \rfloor, 0 < r_2 < r_1 \\
&\vdots \\
r_{n-3} &= r_{n-2} q_{n-1} + r_{n-1} && r_{n-1} = \gcd(a, b) \\
r_{n-2} &= r_{n-1} q_n + r_n && r_n = 0
\end{aligned}$$

Repeated division with remainder. Notice that the sequence of remainders (the r_i) is strictly decreasing, and thus the sequence is finite (algorithm terminates).

Theorem 4.1 (Lamé, 1844). $n < 5 \log_{10} \min(a, b)$ *(i.e., Euclidean algorithm is* $O(\log \min(a, b))$*)*

Let $A_{-2} = 0$, $A_{-1} = 1$, $B_{-2} = 1$, $B_{-1} = 0$ and

$$\begin{aligned}
A_k &= q_k A_{k-1} + A_{k-2} \\
B_k &= q_k B_{k-1} + B_{k-2}
\end{aligned}$$

for $k = 0, 1, \ldots$. We have $A_n = a$ and $B_n = b$ (n from above), and

$$A_k B_{k-1} - B_k A_{k-1} = (-1)^{k-1} \ .$$

Putting $k = n$ yields

$$\begin{aligned}
A_n B_{n-1} - B_n A_{n-1} &= (-1)^{n-1} \\
a(-1)^{n-1} B_{n-1} + b(-1)^n A_{n-1} &= 1 \ .
\end{aligned}$$

Thus, a solution of $ax + by = 1$ is given by

$$x = (-1)^{n-1} B_{n-1}, \quad y = (-1)^n A_{n-1} \ .$$

Computing x, y in this manner is known as the *Extended Euclidean Algorithm.*

4.1.1 Inverses in $\mathbb{F}_m^*$

Consider the linear congruence $ax \equiv 1 \pmod{m}$ given $a, m \in \mathbb{Z}$, $\gcd(a,m) = 1$. We want x such that

$$m \mid ax - 1 \implies ax - 1 = ym \implies ax - my = 1$$

(can be solved using Euclidean Algorithm).

Example 4.1. For $412x \equiv 1 \pmod{317}$ we obtain $x = -10$, so $x \equiv 307 \pmod{317}$ is a solution.

The same algorithm works with $a, m \in \mathbb{F}_q[x]$.

4.2 The Power Algorithm (Binary Exponentiation)

Evaluate $a^n \pmod{m}$ given a, n, m.

Let $n = b_0 2^k + b_1 2^{k-1} + \cdots + b_{k-1}2 + b_k$ be the binary expansion of n, i.e., $b_0 = 1$, $b_i \in \{0,1\}$, $1 \le i \le k$, $k = \lfloor \log_2 n \rfloor$. Given $b_0, \ldots, b_k$, we can evaluate n efficiently ($O(\log_2 n)$) using *Horner's Method*:

$$n = 2(\ldots(2(2b_0 + b_1) + b_2)\cdots + b_{k-1}) + b_k \ .$$

Put

$$\begin{aligned} s_0 &= b_0 \\ s_{i+1} &= 2s_i + b_{i+1} \qquad\qquad (i = 0, 1, \ldots, k-1) \\ s_k &= n \ . \end{aligned}$$

and define $r_i \equiv a^{s_i} \pmod{m}$ for $i = 0, \ldots, k$. Then $r_k \equiv a^{s_k} = a^n$ and we can compute r_{i+1} iteratively from r_i as follows:

$$\begin{aligned} r_{i+1} &\equiv a^{s_{i+1}} \equiv a^{2s_i + b_{i+1}} \equiv (a^{s_i})^2 a^{b_{i+1}} \equiv (r_i)^2 a^{b_{i+1}} \pmod{m} \\ &\equiv \begin{cases} r_i^2 \mod m & \text{if } b_{i+1} = 0 \\ r_i^2 a \mod m & \text{if } b_{i+1} = 1 \end{cases} \ . \end{aligned}$$

This gives us an $O(\log_2 n)$ algorithm to compute a^n. Note that there at most $2\log_2 n$ multiplications, and no operands larger than m^2.

4.3 Euler's ϕ Function

Given m what is $|\mathbb{Z}_m^*|$?

Definition 4.1. Let m be a positive integer. We define

$$\phi(m) = |\{r \in \mathbb{Z} \mid 0 < r < m, \gcd(r,m) = 1\}| \ .$$

Note. $\phi(m) = |\mathbb{Z}_m^*|$, the number of integers between 1 and $m-1$ which have multiplicative inverses modulo m.

Let p be a prime. Then

$$\begin{aligned} \phi(p) &= p - 1 = p^0(p-1) \\ \phi(p^2) &= p^2 - p = p^1(p-1) \\ \phi(p^n) &= p^n - p^{n-1} = p^{n-1}(p-1) \ . \end{aligned}$$

Let $m = m_1 m_2$ be composite. Can we compute $\phi(m)$ if we know $\phi(m_1)$, $\phi(m_2)$, and $\gcd(m_1, m_2) = 1$?

Theorem 4.2 (Chinese Remainder Theorem). *If* $M = m_1 m_2 \dots m_k$ *with* $\gcd(m_i, m_j) = 1$ *for all* $1 \le i, j \le k$ *then there is only one solution* $X \bmod M$ *of*

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_k \pmod{m_k}. \end{aligned}$$

The solution is given by

$$X = \sum_{i=1}^{k} M_i \zeta_i a_i \pmod{M}$$

where $M_i = M/m_i$ *and* $M_i \zeta_i \equiv 1 \pmod{m_i}$.

Example 4.2. CRT for 2 moduli: if $x \equiv a_1 \pmod{m_1}$ and $x \equiv a_2 \pmod{m_2}$, then $x = m_2 t_1 a_1 + m_1 t_2 a_2 \pmod{m_1 m_2}$, where

$$\begin{aligned} m_2 t_1 &\equiv 1 \pmod{m_1} \\ m_1 t_2 &\equiv 1 \pmod{m_2} \end{aligned}$$

Another interpretation: the map

$$\begin{aligned} \phi : \mathbb{Z}_{m_1}^* \times \mathbb{Z}_{m_2}^* \times \cdots \times \mathbb{Z}_{m_k}^* &\to \mathbb{Z}_{m_1 m_2 \dots m_k}^* \\ (a_1, a_2, \dots, a_k) &\mapsto a \end{aligned}$$

where $a \equiv a_i \pmod{m_i}$, $1 \le i \le k$ is an isomorphism.

Theorem 4.3. *If* $\gcd(m_1, m_2) = 1$, *then* $\phi(m_1 m_2) = \phi(m_1)\phi(m_2)$.

Proof. Follows from CRT. □

Corollary 4.4. *If the prime factorization of* m *is given by*

$$m = \prod_{i=1}^{k} p_i^{\alpha_i}, \quad p_i \text{ prime},$$

then

$$\phi(m) = \prod_{i=1}^{k} \phi(p_i^{\alpha_i}) = \prod_{i=1}^{k} p_i^{\alpha_i - 1}(p_i - 1) \ .$$

Example 4.3. $\phi(42) = \phi(2 \times 3 \times 7) = \phi(2)\phi(3)\phi(7) = (1)(2)(6) = 12.$

$\phi(360) = \phi(2^3 3^2 5) = (2^3 - 2^2)(3^2 - 3)(5 - 1) = 4 \cdot 6 \cdot 4 = 96.$

Recall $\mathbb{Z}_m^*$ — a finite group of order $\phi(m)$. If $[a] \in \mathbb{Z}_m^*$, then

$$\begin{aligned} [a]^{\phi(m)} &= e = [1] \\ [a^{\phi(m)}] &= [1] \\ a^{\phi(m)} &\equiv 1 \pmod{m} \end{aligned}$$

Theorem 4.5 (Euler). *If* $\gcd(a, m) = 1$, *then*

$$a^{\phi(m)} \equiv 1 \pmod{m} \ .$$

Theorem 4.6 (Fermat). *If p is prime, then $\phi(p) = p - 1$ and if $p \nmid a$*

$$a^{p-1} \equiv 1 \pmod{p} \ .$$

Note. Contrapositive yields a probabilistic primality test using binary exponentiation:

- If $a^{N-1} \equiv 1 \pmod{N}$ for a few small primes $a \nmid N$, then N is probably prime (base a pseudoprime).
- If $a^{N-1} \not\equiv 1 \pmod{N}$ for any prime $a \nmid N$, then N is composite.

Unfortunately, there are composite numbers (called *Charmichael numbers*) for which $a^{N-1} \equiv 1 \pmod{N}$ for all a with $\gcd(a, n) = 1$. An example is 1729. Thus, this method cannot prove primality.

4.4 Primitive Roots

Definition 4.2. We say that g is a *primitive root* of m if $\mathbb{Z}_m^* = \langle [g] \rangle$, i.e., if $[g]$ is a generator.

Example 4.4. Is $a = 3$ a generator modulo $p = 7$? By tabulating the powers of 3 mod 7 we get

$$3^0 = 1, \quad 3^1 = 3, \quad 3^2 = 2, \quad 3^3 = 6, \quad 3^4 = 4, \quad 3^5 = 5, \quad 3^6 = 1, \quad 3^7 = 3, \ldots$$

(sequence repeats). Since 6 is the smallest power of a yielding 1, 3 is a primitive root modulo 7. 5 is also a primitive root modulo 7. There are no others (e.g. 2 has period 3).

Theorem 4.7. *The only integers which can have primitive roots are $2, 4, p^n, 2p^n$ where $n \in \mathbb{Z}^+$ and p is any odd prime.*

Theorem 4.8. *If m has a primitive root, then it has $\phi(\phi(m))$ primitive roots.*

Computing Primitive Roots

Given a prime p, find a primitive root. Randomly select g and test whether it is a primitive root (probability about 3/8).

When is g a primitive root modulo p? Let $r_1, r_2, \ldots, r_k$ be the distinct prime divisors of $p - 1$. Then g is a primitive root of p if and only if

$$g^{\frac{p-1}{r_i}} \not\equiv 1 \pmod{p} \quad \text{for } i = 1, 2, \ldots, k$$

Example 4.5. $p = 19$. Select $q = 2$. $p - 1 = 18 = 2 \times 3^2$ and

$$2^{(19-1)/2} = 2^9 \equiv 18 \pmod{19}$$
$$2^{(19-1)/3} = 2^6 \equiv 7 \pmod{19} \ .$$

Thus, 2 is a primitive root of 19.

Week 5

Public-Key Cryptography and RSA

5.1 One-way Functions

A function f is said to be a *one-way function* if:

1. $f(x)$ is easy to evaluate for a given x.
2. Given $y = f(x)$, it is computationally infeasible to find x.

Example 5.1. A secure cryptosystem (computationally infeasible to find the key) provides a one-way function. Define $y = f(x) = E_x(P_0)$, where P_0 is a known piece of plaintext. Given P_0 and y (KTA) it should not be easy to compute the key x. We could also use $f(x) = E_x(x)$.

Example 5.2. Let p be a large prime ($\approx 2^{1024}$) and g be a primitive root. Define

$$f(x) \equiv g^x \pmod{p}, \quad 0 < x, f(x) < p \ .$$

This seems to be a one-way function if $p-1$ has at least one large prime divisor. Computing x given $f(x)$ and g is known as the *discrete logarithm problem.*

Best known algorithms:

- Number field sieve — $O(e^{(c+o(1))(\lg n)^{1/3}(\lg\lg n)^{2/3}})$ with $c \approx 1.923$.
- Pohlig-Hellman — $O(\sum e_i(\lg n + \sqrt{p_i}))$ if $p - 1 = p_1^{e_1} \dots p_i^{e_i}$.

Example 5.3. Let G be any (multiplicatively written) group with $|G|$ large and define $f(x) = g^x$. Computing x given $g, f(x)$ is called the discrete logarithm problem in G (the previous example is a special case).

Best known algorithms:

- Generic (Baby-step giant-step, Pollard-rho) — $O(\sqrt{|G|})$
- Pohlig-Hellman — same as above
- Possibly better (even polynomial time) depending on the group.

5.2 Diffie-Hellman Key Exchange

A and B wish to exchange a key for encryption over a public channel in such a way that a tapper cannot determine the key.

Solution — Due to Diffie and Hellman (1976), still used today.

Select a group G for which $n = |G|$ is large and an element g of large order (a generator if possible).

A	Public Channel	B
Select $0 < x_1 < n$ randomly		Select $0 < x_2 < n$ randomly
$y_1 = g^{x_1}$	$y_1 \longrightarrow$	y_1
y_2	$\longleftarrow y_2$	$y_2 = g^{x_2}$
$K = y_2^{x_1}$		$K = y_1^{x_2}$

Notice that

$$y_2^{x_1} = (g^{x_2})^{x_1} = g^{x_2 x_1} = (g^{x_1})^{x_2} = y_1^{x_2},$$

i.e., A and B have the same K at the end of the protocol. Use K for the key (e.g. take 128 high order bits).

Originally presented in $\mathbb{F}_p^*$ (multiplication mod p, g a primitive root modulo p) but works in any group for which the DLP is hard.

Security

The tapper has g, y_1, y_2

- Security is equivalent (provable security) to the Diffie-Hellam problem — given g^{x_1} and g^{x_2}, compute $g^{x_1 x_2}$.
- DHP $\leq_m^P$ DLP (other direction unknown)

5.3 One-way Trapdoor Functions

A function f is said to be a *one-way trapdoor function* if:

1. $f(x)$ is easy to compute for any x.
2. Given $y = f(x)$ it is computationally infeasible to determine x unless certain information used in the design of f is known. When this special information k is known, there exists a function g which is easy to compute such that $x = g(k, y)$.

5.4 Public-Key Cryptography

A *public-key cryptosystem* consists of two functions E_{K_1} and D_{K_2} with the following properties:

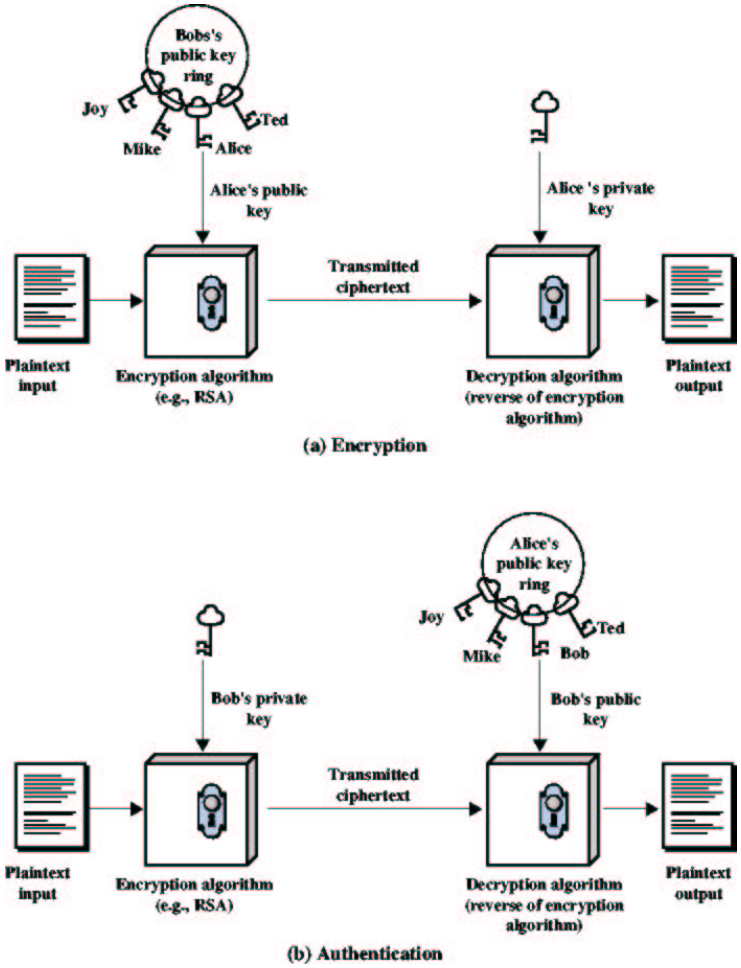
1. $E_{K_1}(M)$ and $D_{K_2}(C)$ are easy to compute when K_1, K_2 are known.
2. $D_{K_2}(E_{K_1}(M)) = M$.
3. Given K_1, E_{K_1}, $C = E_{K_1}(M)$ it is computationally infeasible to find M or K_2.

Note. In a public-key cryptosystem, it is *not* necessary for the key channel to be secure.

Properties 1,2,3 describe E_{K_1} as a one-way trapdoor function. For a signature system (authentication) we require:

4. $E_{K_1}(D_{K_2}(C)) = C$.

Public-key authentication: A computes $S = D_{K_2}(M)$ and B verifies $E_{K_1}(S) = E_{K_1}(D_{K_2}(M)) = M$.



5.5 The RSA Cryptosystem

Named after Ron Rivest, Adi Shamir, and Len Adelman.

The designer:

1. Selects two distinct large primes p and q (each around $2^{512} \approx 10^{155}$)
2. Let $n = pq$, $\phi(n) = (p-1)(q-1)$.
3. Select at random an integer e such that $\gcd(e, \phi(n)) = 1$ and $1 < e < n$.
4. Solves the linear congruence

$$de \equiv 1 \pmod{\phi(n)} \ .$$

5. Keeps d secret and makes n and e public, i.e., the *public key* $K_1 = n, e$ and the *private key* $K_2 = d, p, q$.

Suppose A wants to send a message M to the designer D. We assume that M is an integer and that $M < n$. If $M > n$, block it into less-than-n size blocks.

1. A computes $C \equiv M^e \pmod{n}$ where $0 < C < n$ (e and n are D's public key — everyone knows them)
2. A transmits C.
3. D computes $C^d \pmod{n}$ (d is D's private key — only D knows d)

Note. Encryption and decryption are done using the power algorithm (fast).

Why does this work? We have

$$C^d \equiv (M^e)^d \equiv M^{ed} \pmod{n},$$

but since d is chosen such that $ed \equiv 1 \pmod{\phi(n)}$ we have $ed = k\phi(n) + 1$ for some $k \in \mathbb{Z}$, and

$$M^{ed} \equiv M^{k\phi(n)+1} \equiv MM^{k\phi(n)} \equiv M(M^{\phi(n)})^k \pmod{n} \ .$$

Euler's Theorem states that $a^{\phi(n)} \equiv 1 \pmod{n}$, so we have

$$C^d \equiv M(M^{\phi(n)})^k \equiv M(1)^k \equiv M \pmod{n} \ .$$

Note. We have assumed that $\gcd(M, n) = 1$ in applying Euler's Theorem — works for any M (see assignment).

5.5.1 Security of RSA

Factoring n

If n can be factored, RSA is broken. Other direction unknown (computational security).

Small encryption exponents

Small e are insecure if the same message (or only slightly different), is sent to many entities.

Small decryption exponents

The decryption exponent d must be $> \sqrt[4]{n}/2$ (Wiener 1990).

Improved to $d > n^{0.292}$ by Boneh and Durfee (EUROCRYPT 1999)

Finding d or $\phi(n)$

Knowledge of d or $\phi(n)$ is equivalent to factoring n (see Assignment 3).

Iterative Attack

Let $C_0 = C = f(M)$ (f is the encryption function). Define:

$$C_i = f(C_{i-1}) \ .$$

Stop when $C_m = C_0 \implies M = C_{m-1}$.

This technique is provable equivalent to factoring n under the assumption of the Extended Riemann Hypothesis.

Let

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

where $s = \sigma + ti$ (a complex number). When is $\zeta(s) = 0$?

- Trivial zeros: $s = -2n$
- Non-trivial zeros: Riemann hypothesis states that if s is a non-trivial root, then $\sigma = 1/2$.

Over 1.5 billion non-trivial zeros have been found, and for all of them $\sigma = 1/2$. The Riemann hypothesis has resisted proof since 1853.

The *extended Riemann hypothesis* says that

$$L(s, \chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s}$$

has its non-trivial zeros on the same line ($\sigma = 1/2$), and also remains unproved.

Multiplicative Property

For any messages $1 < M_1, M_2 < n$,

$$(M_1 M_2)^e \equiv M_1^e M_2^e \equiv C_1 C_2 \pmod{n}$$

This property can be used to mount an adaptive chosen ciphertext attack when the victim will decrypt arbitrary ciphertexts for the attacker:

1. The attacker computes $\overline{C} \equiv CX^e \pmod{n}$ (wants the decryption of C)
2. The victim decrypts $\overline{M} \equiv \overline{C}^d \equiv C^d (X^e)^d \equiv MX \pmod{n}$. for the attacker.
3. The attacker recovers M by computing $M \equiv \overline{M} X^{-1} \pmod{n}$.

This attack can be prevented by, for example, imposing a structure on all messages (reject non-conforming decryptions).

5.5.2 Summary of RSA

There is no proof that RSA is secure.

- If n can be factored, then RSA is broken. It is not proven whether factoring is hard (computational security).

- If d or $\phi(n)$ can be found, it can be used to factor n.
- It is not proven that other methods to compute M given C, e, n do not exist, which do not rely on factoring. I.e., it is not known whether breaking RSA is *equivalent* to factoring n (provable security).

Nevertheless, we need to design RSA systems such that $n = pq$ cannot be factored easily.

1. p and q must both be large ($2^{512} \approx 10^{155}$, soon to be $2^{1024} \approx 10^{308}$). This means $n \approx 2^{1024}$ (or 2^{2048}).
2. $|p - q| \gg \sqrt[4]{n}$
3. $p-1$, $q-1$, $p+1$, $q+1$ must all have a large prime factor. There are factoring algorithms which can exploit these properties.

The best general purpose factoring algorithm currently available (the "number field sieve") can factor n in

$$\log n < e^{((\log n)^{1/3}(\log\log n)^{2/3})(1+o(1))} < n$$

operations. Massive parallelism, internet distributed computations, are applicable with this algorithm.

Current factoring record (integer without a special form): RSA-155, a 155-digit RSA modulus was factored in August 1999 (total time — 20 years on a single 450 MHz PC with 64 MB RAM).

Advantages:

1. It seems to be secure.
2. The key size is "small" (two 310 digit numbers).
3. No message expansion.
4. It can be used as a signature scheme.

Disadvantages:

1. It is very slow compared with DES, RIJNDAEL, and other *private key* cryptosystems.
2. Finding keys is fairly expensive.
3. Security is unproven.

Week 6

More Public-Key Cryptosystems

6.1 Quadratic Residues

Definition 6.1. Let m be any integer. We say that a is a *quadratic residue* of m if $\gcd(a, m) = 1$ and there exists some x such that $x^2 \equiv a \pmod{m}$.

Let QR_m denote the set of quadratic residues modulo m.

Definition 6.2. If a is such that $\gcd(a, m) = 1$ and $a \notin QR_m$, we say that a is a *quadratic non-residue* of m.

Let QN_m denote the set of quadratic non-residues of m.
Note. $\mathbb{Z}_m^* = QR_m \cup QR_m$.

Suppose $m = p$, a prime.

Example 6.1. If $p = 7$ we have $QR_7 = \{1, 2, 4\}$ and $QN_7 = \{3, 5, 6\}$.

Theorem 6.1 (Euler's Criterion). *$a \in QR_p$ if and only if $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$.*

Proof. If $a \in QR_p$, then $x^2 \equiv a \pmod{p}$ for some x. Then

$$a^{\frac{p-1}{2}} \equiv (x^2)^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \pmod{p} \quad \text{by Euler's Theorem}$$

Suppose $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ and let g be a primitive root modulo p. There must exist some i such that $g^i \equiv a \pmod{p}$, so

$$g^{i\frac{p-1}{2}} \equiv a^{\frac{p-1}{2}} \equiv 1 \pmod{p} \ .$$

Therefore $g^{i\frac{p-1}{2}} \equiv 1 \pmod{p}$ and i can be even or odd. If i is odd, then $i = 2k+1$ and $i\frac{p-1}{2} = k(p-1) + \frac{p-1}{2}$ and

$$g^{i\frac{p-1}{2}} \equiv g^{k(p-1)} g^{\frac{p-1}{2}} \equiv g^{\frac{p-1}{2}} \equiv -1 \not\equiv 1 \pmod{p} \ .$$

Thus $i = 2k$ and putting $x \equiv g^k \pmod{p}$ we get $a \equiv x^2 \pmod{p}$ and $a \in QR_p$. □

Definition 6.3. Let p be an odd prime. The *Legendre symbol* $\left(\frac{a}{p}\right)$ is defined as:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p \mid a \\ 1 & \text{if } a \in QR_p \\ -1 & \text{if } a \in QN_p \end{cases}$$

Note. $a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$. Eg. $\left(\frac{6}{7}\right) = -1$.

Properties of the Legendre symbol:

1. $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}} = 1$ if $p \equiv 1 \pmod 4$ and -1 if $p \equiv 3 \pmod 4$.
2. $\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$.
3. $\left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$ if $a \equiv b \pmod p$.
4. $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}} = 1$ if $p \equiv \pm 1 \pmod 8$ and -1 if $p \equiv \pm 3 \pmod 8$.
5. $\left(\frac{t^2}{p}\right) = 1$ if $p \not| t$.
6. $\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}\left(\frac{p}{q}\right)$ (Law of Quadratic Reciprocity)

Note. Let g be a primitive root modulo p. Then

$$QR_p = \{g^{2i} \mid i = 1, \ldots, \frac{p-1}{2}\} \quad \text{and} \quad QN_p = \{g^{2i-1} \mid i = 1, \ldots, \frac{p-1}{2}\}$$

Note that $|QR_p| = |QN_p| = \frac{p-1}{2}$.

Example 6.2. Evaluate $\left(\frac{319}{1031}\right)$.

One way is $315^{\frac{1031-1}{2}} \equiv 319^{515} \pmod{1031}$ but quadratic reciprocity yields a better way:

$$\left(\frac{319}{1031}\right) = \left(\frac{11 \cdot 29}{1031}\right) = \left(\frac{11}{1031}\right)\left(\frac{29}{1031}\right) = -\left(\frac{1031}{11}\right)\left(\frac{1031}{29}\right) = -\left(\frac{8}{11}\right)\left(\frac{16}{29}\right) = -\left(\frac{2}{11}\right) = -(-1) = 1$$

The Jacobi Symbol

Definition 6.4. Let Q be an odd integer with prime factorization $Q = \prod q_i^{\alpha_i}$ and let P be an integer. The *Jacobi symbol* $\left(\frac{P}{Q}\right)$ is defined as

$$\left(\frac{P}{Q}\right) = \prod \left(\frac{P}{q_i}\right)^{\alpha_i}$$

where $\left(\frac{P}{q_i}\right)$ is the Legendre symbol.

Note. If Q is prime, then the Jacobi symbol $\left(\frac{P}{Q}\right)$ and the Legendre symbol $\left(\frac{P}{Q}\right)$ are the same.

Note. $\left(\frac{P}{Q}\right) = 1$ with Q composite does *not* necessarily imply that $x^2 \equiv P \pmod Q$ has solutions (would need $x^2 \equiv P \pmod{q_i}$).

Properties of the Jacobi symbol:

1. $\left(\frac{-1}{Q}\right) = (-1)^{\frac{Q-1}{2}}$
2. $\left(\frac{P_1P_2}{Q}\right) = \left(\frac{P_1}{Q}\right)\left(\frac{P_2}{Q}\right)$
3. if $P_1 \equiv P_2 \pmod Q$, then $\left(\frac{P_1}{Q}\right) = \left(\frac{P_2}{Q}\right)$
4. $\left(\frac{2}{Q}\right) = (-1)^{\frac{Q^2-1}{8}} = 1$ if $Q \equiv \pm 1 \pmod 8$ and -1 if $Q \equiv \pm 3 \pmod 8$
5. $\left(\frac{P}{Q}\right) = (-1)^{\frac{P-1}{2}\frac{Q-1}{2}}\left(\frac{Q}{P}\right)$ (for P odd)

Example 6.3. Evaluate $\left(\frac{319}{1031}\right)$.

If we treat this as a Jacobi symbol, we don't need to do any factoring.

$$\left(\frac{319}{1031}\right) = -\left(\frac{1031}{319}\right) = -\left(\frac{74}{319}\right) = -\left(\frac{2}{319}\right)\left(\frac{37}{319}\right) = -\left(\frac{37}{319}\right) = -\left(\frac{319}{37}\right) = -\left(\frac{23}{37}\right) = -\left(\frac{37}{23}\right) = -\left(\frac{14}{23}\right)$$
$$= -\left(\frac{7}{23}\right) = \left(\frac{23}{7}\right) = \left(\frac{2}{7}\right) = 1$$

6.1.1 Square roots modulo p

Given a prime p and an a such that $\left(\frac{a}{p}\right) = 1$, solve the congruence $x^2 \equiv a \pmod{p}$.

Case 1

Suppose $p \equiv -1 \pmod{4}$. Then

$$\left(a^{\frac{p+1}{4}}\right)^2 \equiv a^{\frac{p+1}{2}} \equiv aa^{\frac{p-1}{2}} \equiv a \pmod{p} \ .$$

Hence $x \equiv \pm a^{\frac{p+1}{4}} \pmod{p}$. Eg. $p = 1031 \equiv -1 \pmod{4}$, $a = 319$. We geave $319^{1032/4} \equiv 319^{258} \equiv 230 \pmod{1031}$.

Case 2

Suppose $p \equiv 5 \pmod{8}$. Since $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, then $a^{\frac{p-1}{4}} \equiv \pm 1 \pmod{p}$. If $a^{\frac{p-1}{4}} \equiv 1 \pmod{p}$, then

$$\left(a^{\frac{p+3}{8}}\right)^2 \equiv a^{\frac{p+3}{4}} \equiv aa^{\frac{p-1}{4}} \equiv a \pmod{p}$$

and thus $x \equiv \pm a^{\frac{p+3}{8}} \pmod{p}$. If $a^{\frac{p-1}{4}} \equiv -1 \pmod{p}$, then

$$\left(\frac{1}{2}(4a)^{\frac{p+3}{8}}\right)^2 \equiv \frac{1}{4}(4a)^{\frac{p+3}{4}} \equiv 4^{\frac{p+3}{4}-1}a^{\frac{p-1}{4}}a \equiv -4^{\frac{p-1}{4}}a \equiv -1\left(2^{\frac{p-1}{2}}\right)a \equiv a \pmod{p}$$

since $\left(\frac{2}{p}\right) = -1$ (because $p \equiv 5 \pmod{8}$). In this case $x \equiv \pm\frac{1}{2}(4a)^{\frac{p+3}{8}} \pmod{p}$.

Case 3

(See Handbook of Applied Cryptography, Section 3.5.1) Suppose $p \equiv 1 \pmod{8}$. The following randomized algorithm will find the square roots in expected time $O(\lg^4 p)$.

1. Select a random integer b with $1 \le b \le q-1$ until $\left(\frac{b}{p}\right) = -1$.
2. Compute s such that $p - 1 = 2^s t$, t odd
3. Set $c \equiv b^t \pmod{p}$ and $r \equiv a^{\frac{t+1}{2}} \pmod{p}$.
4. For $i = 1, \ldots, s-1$ do
   (a) Compute $d \equiv (r^2 a^{-1})^{s^{s-i-1}} \pmod{p}$.
   (b) If $d \equiv -1 \pmod{p}$, set $r \equiv rc \pmod{p}$.
   (c) Set $c \equiv c^2 \pmod{p}$.
5. Return $(r, -r)$

Finding the quadratic non-residue (Step 1) is the randomized step of the algorithm. Under the Extended Riemann hypotheis this step can be done deterministically in polynomial time.

6.2 The Rabin-Williams PKC

Public-key encryption scheme provably equivalent to integer factorization (Rabin, Williams 1980).

Modification of Rabin's scheme (see Assignment) with unique decryption.

Lemma 6.2. *Let $n = pq$ with $p \equiv q \equiv -1 \pmod 4$. If $\left(\frac{M}{n}\right) = 1$, then*

$$M^{\phi(n)/4} \equiv \pm 1 \pmod n$$

Proof. $\left(\frac{M}{pq}\right) = 1 \Longrightarrow \left(\frac{M}{p}\right) = \left(\frac{M}{q}\right)$. If $\left(\frac{M}{p}\right) = 1$, then

$$M^{\frac{p-1}{2}} \equiv 1 \pmod p$$
$$M^{\frac{p-1}{2}\frac{q-1}{2}} \equiv 1 \pmod p \ .$$

Similarly,

$$M^{\frac{q-1}{2}\frac{p-1}{2}} \equiv 1 \pmod q$$

and by the CRT we have $M^{\phi(n)/4} \equiv 1 \pmod n$.

If $\left(\frac{M}{p}\right) = -1$, then we use the fact that $(-1)^{(p-1)/2} \equiv -1 \pmod p$ when $p \equiv -1 \pmod 4$ to argue that $M^{\phi(n)/4} \equiv -1 \pmod n$. □

Key Generation

Select large primes p, q with $p \equiv 3 \pmod 8$, $q \equiv 7 \pmod 8$, and put $n = pq$.

Select at random e such that $1 < e < n$ and $\gcd(e, \phi(n)) = 1$.

Solve $ed \equiv m \pmod{\phi(n)}$ where $m = (\phi(n)/4 + 1)/2$.

Public key: $\{n, e\}$ Private key: $\{d\}$

Encryption and Decryption

Define $\mathcal{M} = \{M \mid (2(2M+1) < n \text{ and } \left(\frac{2M+1}{n}\right) = -1) \text{ or } (4(2M+1) < n \text{ and } \left(\frac{2M+1}{n}\right) = 1)\}$.

Theorem 6.3. *$|\mathcal{M}| = 3/16\phi(n) - t$ and $t < 1/2\sqrt{n}\log n + 5/4$ (i.e., $|\mathcal{M}| \in O(n)$).*

For $M \in \mathcal{M}$ define:

$$E_1(M) = \begin{cases} 4(2M+1) & \text{if } \left(\frac{2M+1}{n}\right) = 1 \\ 2(2M+1) & \text{if } \left(\frac{2M+1}{n}\right) = -1 \end{cases} \quad \left(\text{note } \left(\frac{E_1(M)}{n}\right) = 1\right)$$

$$E_2(N) \equiv N^{2e} \pmod n \quad (0 < E_2(N) < n \text{ and } N \in \mathbb{Z}),$$

$$D_2(K) \equiv K^d \pmod n \quad (0 < D_2(K) < n),$$

$$D_1(L) = \begin{cases} (L/4 - 1)/2 & \text{if } L \equiv 0 \pmod 4 \\ ((n-L)/4 - 1)/2 & \text{if } L \equiv 1 \pmod 4 \\ (L/2 - 1)/2 & \text{if } L \equiv 2 \pmod 4 \\ ((n-L)/2 - 1)/2 & \text{if } L \equiv 3 \pmod 4 \end{cases}$$

To encrypt $M \in \mathcal{M}$, the sender computes $C = E_2(E_1(M))$.

To decrypt C, the receiver computes $D_1(D_2(C)) = M$.

Theorem 6.4. *If $M \in \mathcal{M}$ then $D_1 D_2 E_2 E_1(M) = M$.*

Proof. We have:

$$N = E_1(M) \quad \text{with } 2 \mid N,\ 0 < N < n, \text{ and } \left(\frac{N}{n}\right) = 1$$

$$L = D_2 E_2(N) \equiv N^{2ed} \equiv N^{2m} \equiv N^{\phi(n)/4+1} \equiv \pm N \pmod{n} \quad \text{with } 0 < L < n \text{ and } n \equiv 1 \pmod{4}$$

Thus, if L is even, then $L = N$ and if L is odd, then $L = n - N$.

If $L \equiv 0 \pmod 4$, then $(2M+1) = N/4 = L/4 \Longrightarrow M = (L/4 - 1)/2 = D_1(L)$.

If $L \equiv 1 \pmod 4$, then $2M + 1 = (n - L)/4 \Longrightarrow M = ((n - L)/4 - 1)/2 = D_1(L)$.

If $L \equiv 2 \pmod 4$, then $2M + 1 = L/2 \Longrightarrow M = (L/2 - 1)/2 = D_1(L)$

If $L \equiv 3 \pmod 4$, then $2M + 1 = (n - L)/2 \Longrightarrow M = D_1(L)$. □

We will now show that breaking the encryption scheme is equivalent in difficulty to factoring n.

Lemma 6.5. *If n is given as above, then for any $X \in \mathbb{Z}$ there exists $Y \in \mathbb{Z}$ such that $X^2 \equiv Y^2 \pmod n$ and $\left(\frac{Y}{n}\right) = -\left(\frac{X}{n}\right)$.*

Proof. $\left(\frac{-X}{p}\right) = \left(\frac{-1}{p}\right)\left(\frac{X}{p}\right) = -\left(\frac{X}{p}\right)$. Let

$$Y \equiv -X \pmod p, \quad Y \equiv X \pmod q \ .$$

Then $Y^2 \equiv X^2 \pmod n$ and

$$\left(\frac{Y}{n}\right) = \left(\frac{Y}{p}\right)\left(\frac{Y}{q}\right) = \left(\frac{-X}{p}\right)\left(\frac{X}{q}\right) = -\left(\frac{X}{n}\right) \ .$$

□

Lemma 6.6. *If $K = E(M)$ (here $E = E_2 E_1$), then there exists X_1, X_2 such that $X_1 \neq X_2$, $0 < X_1, X_2 < n$, $X_1^2 \equiv X_2^2 \equiv K \pmod n$ and $\left(\frac{X_1}{n}\right) = \left(\frac{X_2}{n}\right) = -1$.*

Proof. Let $N = E_1(M)$ and $Y \equiv N^e \pmod n$. We have $K \equiv (N^e)^2 \equiv Y^2 \pmod n$ and since $\left(\frac{N}{n}\right) = 1 \Longrightarrow \left(\frac{Y}{n}\right) = 1$. By Lemma 6.5 there exists an X such that $X^2 \equiv Y^2 \equiv K \pmod n$ and $\left(\frac{X}{n}\right) = -1$. Let $X_1 \equiv X \pmod n$, $0 < X_1 < n$, and $X_2 = n - X_1$. □

Put $\mathcal{X} = \{X \mid X^2 \equiv E(M) \pmod n, M \in \mathcal{M}, \left(\frac{X}{n}\right) = -1, 0 < X < n\}$. Then $|X| \geq 2|\mathcal{M}|$ by Lemma 6.6. If we select at random a value of X such that $\left(\frac{X}{n}\right) = -1$ and $0 < X < n$ (there are $\phi(n)/2$ such X values) then the probability that there exists an $M \in \mathcal{M}$ such that $X^2 \equiv E(M) \pmod n$ is about $3/4$.

If F is an algorithm which decrypts $1/k$ of all possible ciphertexts, then we see that we can select at random a value of X $(0 < X < n)$ with $\left(\frac{X}{n}\right) = -1$ such that $E(M) \equiv K \equiv X^2 \pmod n$ for some $M \in \mathcal{M}$ and $F(K) = M$ with probability about $\frac{3}{4k}$. We expect to conduct about $4k/3$ trials before such an example is found. Put $Y \equiv E_1(M)^e \equiv E_1(F(K))^e \pmod n$. Then

$$Y^2 \equiv X^2 \pmod n \text{ and } \left(\frac{Y}{n}\right) = 1, \left(\frac{X}{n}\right) = -1$$

and $n = pq \mid X^2 - Y^2 \Longrightarrow pq \mid (X - Y)(X + Y)$. Now:

- If $pq \mid X - Y$, then $X \equiv Y \pmod{pq}$, and $\left(\frac{X}{n}\right) = \left(\frac{Y}{n}\right)$, a contradiction.
- If $pq \mid X + Y$, then $X \equiv -Y \pmod{pq}$, and $\left(\frac{X}{n}\right) = \left(\frac{-Y}{n}\right) = \left(\frac{Y}{n}\right)$, a contradiction.

Hence, $\gcd(X - Y, n) = p, q$, i.e., we can factor n.

6.3 The El Gamal PKC

It is important to have other public-key cryptosystems whose security relies on other hard problems. RSA relies on factoring. An alternative PKC is due to El Gamal.

A produces her public and private keys as follows:

1. Selects a group G for which $n = |G|$ is large and an element g of large order (a generator if possible). Orignially proposed in $G = \mathbb{F}_p^*$.
2. Computes $y = g^x$ where $0 < a < n$.

Public key: $\{G, g, y\}$
Private key: $\{x\}$

B sends a message M to A as follows:

1. B represents the message M as an element $M \in G$.
2. B selects a random k, $0 < k < n$.
3. B computes $K = y^k$
4. B sends $\{C_1, C_2\}$ to A where
$$C_1 = g^k, \quad C_2 = KM$$

A decrypts as follows:

1. Use the private key x to compute $C_1^x = (g^k)^x = g^{kx} = (g^x)^k = y^k = K$.
2. Solve $C_2 = KM$ for M by computing K^{-1} in G.

Clearly, this PKC is very similar to Diffie-Hellman key exchange (using the common key to encrypt $M \in G$), and the security results are the same. In particular, any large group with a hard DLP is suitable.

Disadvantages:

1. Increased bandwidth — the communication channel must be twice as wide as the message being sent.
2. Twice as much computational work for encrypting and decrypting (as RSA).
3. A new random number, k, must be generated for each message.

Other PKC's

Merkle-Hellman — subset sum problem (NP-complete). First concrete realization of a PKC, but insecure

Chor-Rivest — secure subset-sum based PKC

McEliece — decoding linear error-correcting codes

XTR - subgroup of $F_{p^6}^*$

NTRU - shortest vector in a lattice

Week 7

Probabilistic Public-Key Cryptography

7.1 Semantic Security

Two disadvantages of deterministic public-key cryptographic algorithms are:

1. Having the same message always encrypt to the same ciphertext
2. Leakage of information. For example, in RSA $C \equiv M^e \pmod{n}$, implying that

$$\left(\frac{C}{n}\right) = \left(\frac{M}{n}\right)^e = \left(\frac{M}{n}\right)$$

since e is odd. Thus, one bit of information about the message is leaked (namely the value of the Jacobi symbol $\left(\frac{M}{n}\right)$.

Probabilistic encryption utilizes randomness to attain a provable and very strong level of security. There are two strong notions of security that we can strive to achieve.

Definition 7.1. A public-key encryption scheme is said to be *polynomially secure* if no passive adversary can in expected polynomial time select two plaintext messages M_1 and M_1 and then correctly distinguish between encryptions of M_1 and M_2 with probability significantly greater than 1/2.

Definition 7.2. A public-key encryption scheme is said to be *semantically secure* if for all probability distributions over the message space, whatever can be computed by a passive adversary in expected polynomial time about the plaintext given the ciphertext can also be computed in expected polynomial time without the ciphertext.

Intuitively, semantic security is a weaker version of perfect security — an adversary with polynomially-bounded computational resources (as opposed to infinite in perfect security) can learn nothing about the plaintext from the ciphertext.

Theorem 7.1. *A public-key encryption scheme is semantically secure if and only if it is polynomially secure.*

7.2 The Goldwasser-Micali PKC

Achieves semantic security assuming the intractability of the QRP.

Definition 7.3. *Quadratic Residuosity Problem* (QRP) — given an odd composite integer n and any a such that $\left(\frac{a}{n}\right) = 1$, determine whether $a \in QR_n$.

Note. $QRP \leq_m^P FACTORING$, since $a \in QR_n$ if and only if $\left(\frac{a}{p}\right) = 1$ for all primes $p \mid n$. Equivalence is believed, but unproved.

A produces her public and private key as follows:

1. Selects two large random distinct primes p, q
2. Computes $n = pq$
3. Selects y such that $\left(\frac{y}{p}\right) = \left(\frac{y}{q}\right) = -1$ (y is a *pseudosquare* modulo $n = pq$ since $\left(\frac{y}{n}\right) = 1$ but $\left(\frac{y}{p}\right) = -1$ implies that y is not an integer square)

Public key: $\{n, y\}$
Private key: $\{p, q\}$

To encrypt a message M and send to A, B:

1. Represents M as a bit-string $(m_1 m_2 \dots m_t)$ $(m_i = 0, 1)$
2. For $i = 1, \dots, t$:
   (a) Pick $r \in \mathbb{Z}$ such that $1 < r < n$ at random
   (b) Put $c_i \equiv y^{m_i} r^2 \pmod{n}$ with $0 < c_i < n$
3. Send $C = (c_1 c_2 \dots c_t)$ to A

To decrypt, A:

1. for $i = 1, \dots, t$:
   (a) Compute the Legendre symbol $e_i = \left(\frac{c_i}{p}\right)$
   (b) $m_i = (1 - e_i)/2$
2. $M = (m_1 m_2 \dots m_t)$

Proof that decryption is correct. For any i with $1 \leq i \leq t$ we have $c_i \equiv y^{m_i} r^2 \pmod{n}$. Thus

$$e_i = \left(\frac{c_i}{p}\right) = \left(\frac{y^{m_1} r^2}{p}\right) = \left(\frac{y^{m_i}}{p}\right) = \left(\frac{y}{p}\right)^{m_i} = (-1)^{m_i}$$

Thus, if $e_i = 1$ then $m_i = 0$ and if $e_i = -1$ then $m_i = 1$. □

Security

Since r is selected at random, r^2 is a random quadratic residue modulo n and thus yr^2 is a random pseudosquare modulo n. The cryptanalyst only sees a sequence of r^2 or yr^2 (quadratic residues and pseudosquares), and as the QRP is hard, he cannot distinguish which one from the other.

Major disadvantage — very large message expansion ($\lg^2 n$ bits required as opposed to $\lg n$ for RSA)

7.3 The Blum-Goldwasser PKC

Efficient probabilistic technique, semantically secure assuming the intractability of integer factorization. Smaller message expansion — only $\leq \lfloor \lg n \rfloor$ additional bits.

Idea: a pseudorandom bit stream (from the Blum-Blum-Shub pseudorandom number generator) is XORed with the plaintext. The private key is used to recover the random seed used by the sender to initialize the PRNG.

A creates her public key as follows:

1. Selects two large and distinct primes p, q with $p \equiv q \equiv 3 \pmod 4$.
2. Computes $n = pq$ (n is a *Blum integer*)
3. Finds a and b such that $ap + bq = 1$ with $a, b \in \mathbb{Z}$.

Public key: $\{n\}$
Private key: $\{p, q, a, b\}$

B encrypts M to send to A as follows:

1. Let $k = \lfloor \lg n \rfloor$ and $h = \lfloor \lg k \rfloor \geq 1$. Represent M as a string $M = (m_1 m_2 \dots m_t)$ of length t where each m_i is a binary string of length h.
2. Select a seed x_0 which is a random quadratic residue modulo n (simply select a random $r < n$ and put $x_0 \equiv r^2 \pmod n$).
3. For $i = 1, \dots, t$:
   (a) Compute $x_i \equiv x_{i-1}^2 \pmod n$.
   (b) Let p_i be the least h significant bits of x_i.
   (c) Compute $c_i = m_i \oplus p_i$.
4. Compute $x_{t+1} \equiv x_t^2 \pmod n$.
5. Send $C = (c_1 c_2 \dots c_t, x_{t+1})$ to A.

Note. Only $\lfloor \lg x_{t+1} \rfloor \leq \lfloor \lg n \rfloor$ additional bits transmitted.

A decrypts M from C as follows:

1. Compute
$$d_1 \equiv \left(\frac{p+1}{4}\right)^{t+1} \pmod{p-1}, \quad d_2 \equiv \left(\frac{q+1}{4}\right)^{t+1} \pmod{q-1}$$
2. Compute $u \equiv x_{t+1}^{d_1} \pmod p$ and $v \equiv x_{t+1}^{d_2} \pmod q$.
3. Compute $x_0 \equiv vap + ubq \pmod n$.
4. For $i = 1, \dots, t$:
   (a) Compute $x_i \equiv x_{i-1}^2 \pmod n$.
   (b) Let p_i be the h least significant bits of x_i.
   (c) Compute $m_i = p_i \oplus c_i$.
5. $M = (m_1 m_2 \dots m_t)$.

Proof that decryption is correct. Since $x_t \in QR_n$, we have $x_t \in QR_p \Longrightarrow x_t^{\frac{p-1}{2}} \equiv 1 \pmod{p}$. Thus

$$x_{t+1}^{\frac{p+1}{4}} \equiv (x_t^2)^{\frac{p+1}{4}} \equiv x_t^{\frac{p+1}{2}} \equiv x_t^{\frac{p-1}{2}} x_t \equiv x_t \pmod{p} \ .$$

Similarly, $x_t^{\frac{p+1}{4}} \equiv x_{t-1} \pmod{p}$, and repeating this argument yields

$$u \equiv x_{t+1}^{d_1} \equiv x_0 \pmod{p}, \quad v \equiv x_{t+1}^{d_2} \equiv x_0 \pmod{q} \ .$$

By the CRT we get

$$vap + ubq \equiv x_0 \pmod{n},$$

and thus A creates the same random seed x_0 used by B to encrypt. Hence, A can now decrypt C. □

Security

Note that any method that breaks the scheme must reveal the parity bit of the x_i (the key).

Theorem 7.2. *Let A_n be an algorithm which given any $x \in QR_n$ returns the parity bit of y where $y^2 \equiv x \pmod{n}$ and $y \in QR_n$. Then A_n can be used to solve the QRP for any $[a] \in \mathbb{Z}_n^*$ with $\left(\frac{a}{n}\right) = 1$.*

Proof. Suppose we wish to solve the QRP for some $[a] \in \mathbb{Z}_n^*$. We first determine $x \equiv a^2 \pmod{n}$. We apply A_n to x to get $b = A_n(x)$. Now b is the parity bit of some y where $y^2 \equiv x \pmod{n}$ and $y \in QR_n$. We know $y^2 \equiv a^2 \pmod{n} \Longrightarrow n = pq \,|\, (y-a)(y+a)$. Suppose $p \,|\, y-a$ and $q \,|\, y+a$. Then

$$p \,|\, y-a \Longrightarrow y \equiv a \pmod{p} \Longrightarrow 1 = \left(\frac{y}{p}\right) = \left(\frac{a}{p}\right)$$

and similarly

$$q \,|\, y+a \Longrightarrow y \equiv -a \pmod{q} \Longrightarrow 1 = \left(\frac{y}{q}\right) = \left(\frac{-a}{q}\right) = -\left(\frac{a}{q}\right)$$

and thus $\left(\frac{a}{pq}\right) = \left(\frac{a}{n}\right) = -1$, which is a contradiction. Hence $y \equiv \pm a \pmod{n}$.

- If $y \equiv a \pmod{n}$, then b is the parity bit of a and $a \in QR_n$.
- If $y \equiv -a \pmod{n}$, then $y = n-1$ and b is the parity bit of y and is *not* the parity bit of a (since n is odd).

Thus, if the parity bit of a equals b, then $a \in QR_n$ and if it does not equal b, then $a \notin QR_n$. □

Disadvantage: scheme is vulnerable to a chosen ciphertext attack (in an analogous manner to the Rabin PKC). This can be overcome by adding prescribed structure to each message, but the security proofs are then no longer valid.

Advantage: fast (even compared with RSA).

Week 8

Data Integrity and Digital Signatures

8.1 Hash Functions

Definition 8.1. A *hash function* is a function H which has at least the following properties:

- compression — H maps an input x of arbitrary bitlength to an output $H(x)$ of fixed bitlength,
- ease of computation

Idea: from a given message M, generate a small fixed-length *digital fingerprint* or *message digest* $H(M)$ with the following properties:

1. preimage resistance (one-way) — given any hash value x, it is computationally infeasible to find *any* input M for which $H(M) = x$.
2. second-image resistance (weak collision resistance) — given M, it is computationally infeasible to find $M' \neq M$ with $H(M) = H(M')$.
3. collision resistance (strong collision resistance) — it is computationally infeasible to find two distinct inputs M and M' such that $H(M) = H(M')$.

A message digest provides *data integrity*:

- Send the message M together with $S = H(M)$ (the pair (M, S), or S alone, must be encrypted). The receiver independently computes $H(M)$ and compares with S. If they match, then with very high probability M has not been modified (similar to using a checksum).
- Properties 1,2,3 guarantee that it is computationally infeasible to effect undetectable malicious modifications when $H(M)$ is encrypted (checksums do *not* provide this property).

Examples of hash functions:

- MD5 — 128-bit hash length, developed by Rivest. Collisions can be found in 24 days using a special-purpose machine costing 10 million (van Oorschot and Wiener, Second ACM conference on Computer and Communication Security, 1994). Broken Crypto 2004.
- SHA-1 — 160-bit hash length, developed by NIST in 1993 (FIPS 180 and FIPS 181). No known weaknesses (but not all design criteria is public).

- RIPEMD-160 — 160-bit hash length, developed by the European RACE Integrity Primitives Evaluation (RIPE) project. Broken Crypto 2004.

Collisions on an n-bit hash function can be found in $O(2^{n/2})$ attempts due to the birthday paradox. Thus, at most $n/2$ bits of security are possible.

Eg. 1024-bit RSA provides 80 bits of security, so it should be paired with a 160-bit hash function and an 80-bit block cipher (so that all three components equally strong).

Note. Security levels (NIST recommendations):

RSA modulus (in bits)	1024	2048	3072	8192	15360
Hash size (in bits)	160	224	256	384	512
Security level (in bits)	80	112	128	192	256

8.2 Message Authentication Codes

Similar to hash functions, but a secret key K is required as a parameter. MACs are also known as cryptographic checksums.

In addition to data integrity, MACs provide *message authentication* — only entities possessing the secret key K can generate MACs.

A secure block cipher (satisfying additional statistical properties) can be used to generate MACs. Two methods are:

1. CBC-MAC: Encrypt the message using cipher block chaining. The last cipher block (whose bits are dependent on all the key bits and all message bits) is the MAC.
2. CFB-MAC: Same as CBC-MAC, but use cipher feedback mode to generate the MAC.

A CBC-MAC using DES appears in FIPS 113 and forms the ANSI X9.17 standard.

Note. Non-repudiation of data origin is not provided — *any* party possessing K can generate MACs.

8.3 Digital Signatures

A *digital signature* is a simple means by which the recipient of a message can authenticate the identity of the sender. It should have two properties:

1. Only the sender can produce his signature.
2. *Anyone*, including an arbitrator, should be easily able to verify the validity of the signature.

Note. This is *different* from a MAC — both sender and receiver can generate MACs (eg. using DES).

8.3.1 Authentication Without Secrecy

A sender wishes to send M and a signature indicating that he produced M. If the sender uses a public-key cryptosystem which is also a signature system, he can sign his message.

Suppose A is the sender. She has a decryption scheme D_A and an encryption scheme E_A. She creates her signature by computing

$$S = D_A(M)$$

and sends $\langle M, A, S\rangle$. Anyone who receives $\langle M, A, S\rangle$ can find E_A (because it is public) and verify the message by computing $E_A(S)$ and comparing to M. If the message is authentic, then

$$E_A(S) = E_A(D_A(M)) = M \ .$$

This works with RSA because it is a signature system:

$$\begin{aligned} S = D_A(M) &\equiv M^{d_A} \pmod{n_A} \\ E_A(S) = E_A(D_A(M)) &\equiv (M^{d_A})^{e_A} \pmod{n_A} \\ &\equiv (M^{e_A})^{d_A} \pmod{n_A} \\ &\equiv M \ . \end{aligned}$$

8.3.2 Authentication With Secrecy

Problem: A wants to send a *secret* message M to B. B wants to be sure that A sent M.

Solution: A calculates $S = D_A(M)$ (S is the signature). He then uses B's encryption system to compute $E_B(S, M) = F$ and sends $\langle F, A\rangle$ to B.

B, on receiving $\langle F, A\rangle$ computes

$$D_B(F) = D_B(E_B(S, M)) = S, M$$

and then verifies that

$$E_A(S) = M \ .$$

8.3.3 Impersonation

Problem: C can send a message to B in such a way that B thinks it came from A. C knows E_A, so C can select some L and compute $E_A(L)$. C then sends $\langle E_A(L), L\rangle$ to B. B assumes that L is the signature, evaluates $E_A(L)$, and accepts $E_A(L)$ as the message.

Language redundancy usually foils this attack, since $E_A(L)$ will normally not be coherent English text. However, if A is supposed to be sending random information, this will be a problem.

Solution: A sends $\langle M, D_A(H(M))\rangle$, where H is a one-way function (cryptographically secure hash function is best). B computes $G = E_A(D_A(H(M)))$ and $H(M)$, and checks that they are the same. Only A could send M.

Suppose C attempts to impersonate A as above. If C randomly chooses L, computes $X = H^{-1}(E_A(L))$, and sends $\langle X, L\rangle$ to B, then he will be successful since B will compute $E_A(L)$ and verify that this is equal to

$$H(X) = H(H^{-1}(E_A(L))) = E_A(L) \ .$$

However, if H is a one-way function, C cannot find a suitable X, and therefore will not be able to impersonate A.

Note. Signing $H(M)$ instead of M also results in faster signature generation if M is long.

When a cryptographic hash function is used for signature generation, it should be considered as a fixed part of the protocol (i.e., an adversary should not be able to substitute it with a cryptographically weak hash function).

Examples:

- El Gamal (randomized, security based on DLP in a finite abelian group)
- DSA (variation of El Gamal in $\mathbb{F}_p^*$ with short signatures)
- Feige/Fiat/Shamir (security based on computing square roots modulo $n = pq$)
- Guillou/Quisquater (security based on extracting eth roots modulo $n = pq$)

8.4 The El Gamal Signature Scheme

The El Gamal signature scheme is a variation of the El Gamal PKC. Security considerations are the same.

A cryptographic hash function $H : G \mapsto \mathbb{Z}_n$ is required (computes hash values of the binary representation of a group element).

A produces her public and private keys as follows:

1. Selects a group G for which $n = |G|$ is large and an element g of large order.
2. Randomly selects x such that $0 < x < n$ and computes $y = g^x$.

Public key: $\{G, n, g, y\}$
Private key: $\{x\}$
Note. The order of the group G must be known (not tractable for all groups G).

A signs message M as follows:

1. A selects a random integer k with $0 < k < n$ and $\gcd(k, n) = 1$.
2. A computes $r = g^k$
3. A computes $s = k^{-1}[H(M) - xH(r)] \pmod{n}$.
4. A's signature is the pair (r, s).

B verifies A's signature as follows:

1. B obtains A's authentic public key $\{G, n, g, y\}$.
2. B computes $v_1 = y^{H(r)} r^s$ and $v_2 = g^{H(M)}$.
3. B accepts if and only if $v_1 = v_2$.

Proof of correctness. If the signature (r, s) on message M is valid, then

$$\begin{aligned} v_1 &= y^{H(r)} r^s \\ &= (g^x)^{H(r)} (g^k)^{k^{-1}[H(M) - xH(r)]} \\ &= g^{xH(r)} g^{H(M) - xH(r)} \\ &= g^{xH(r) + H(M) - xH(r)} \\ &= g^{H(M)} \\ &= v_2 \end{aligned}$$

□

8.5 The Digital Signature Algorithm (DSA)

Invented by NIST (National Institute for Standards and Technology) in 1991 and adapted as a standard (Digital Signature Standard) in Dec. 1994.

Variation of El Gamal signatures — similar security characteristics.

Let H be a cryptographically secure hash function that maps bit strings to $\mathbb{Z}_q$ for some integer q. The DSS specifies that SHA-1 be used.

A produces her public and private keys as follows:

1. Selects a 512-bit prime p and a 160-bit prime q such that $q \mid p-1$.
2. Selects g, a primitive root modulo p
3. Computes $h \equiv g^{(p-1)/q} \pmod{p}$, $0 < h < p$. Note that $h^q \equiv 1 \pmod{p}$, and if $a \equiv b \pmod{q}$, then $h^a \equiv h^b \pmod{p}$.
4. Randomly selects $x \in \mathbb{Z}$ with $0 < x < q$ and computes $y \equiv h^x \pmod{p}$

Public key: $\{p, q, h, y\}$
Private key: $\{x\}$

A signs message M as follows:

1. A selects a random integer k with $0 < k < q$.
2. A computes $r \equiv (h^k \bmod p) \pmod{q}$, $0 < r < q$.
3. A computes $s \equiv k^{-1}(H(M) + xr) \pmod{q}$.
4. A's signature is the pair $\{r, s\}$ (320 bits)

B verifies A's signature as follows:

1. B obtains A's authentic public key $\{p, q, h, y\}$.
2. B computes $u_1 \equiv H(M)s^{-1} \pmod{q}$, $u_2 \equiv rs^{-1} \pmod{q}$, and $v \equiv (h^{u_1} y^{u_2} \bmod p) \pmod{q}$, $0 < v < q$.
3. B accepts if and only if $v = r$.

Proof of Correctness. We note that $k \equiv s^{-1}(H(M) + xr) \pmod{q}$ and

$$\begin{aligned} h^{u_1} y^{u_2} &\equiv h^{H(M)s^{-1}} y^{rs^{-1}} \pmod{p} \\ &\equiv h^{H(M)s^{-1}} h^{xrs^{-1}} \pmod{p} \\ &\equiv h^{s^{-1}(H(M)+xr)} \pmod{p} \\ &\equiv h^k \pmod{p} \end{aligned}$$

Hence $(h^{u_1} h^{u_2} \bmod p) \equiv r \pmod{q}$ and $v = r$. □

Note. We have a small signature (320 bits) but computations are done modulo a 512-bit prime. Security is based on the belief that solving the DLP in $\langle [h] \rangle \in \mathbb{F}_p^*$ is hard.

8.6 Elliptic Curves

The most promising implementations of El Gamal and DSA signatures is to use for the group G the set of points on an elliptic curve defined over a finite field. The corresponding discrete logarithm problem appears to be very difficult (best known algorithms have exponential complexity).

Definition 8.2. Let $\mathbb{F}$ be a field. An *elliptic curve* E over $\mathbb{F}$ is a curve given by an equation of the form

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \quad a_i \in \mathbb{F} \ .$$

$E(\mathbb{F})$ denotes the set of points $(x, y) \in \mathbb{F}^2$ that satisfy this equation together with a "point at infinity" denoted by O.

If $char(\mathbb{F}) \neq 2, 3$, the equation of the curve can be transformed to

$$Y^2 = X^3 + aX + b, \quad a, b \in \mathbb{F}$$

via a change of variables. If $char(\mathbb{F}) = 2$, the equation transforms to

$$Y^2 + cY = X^3 + aX + b \quad \text{or} \quad Y^2 + XY = X^3 + aX^2 + b$$

and if $char(\mathbb{F}) = 3$

$$Y^2 = X^3 + aX^2 + bX + c \ .$$

8.6.1 The Group Law

$E(\mathbb{F})$ forms an abelian group under an operation called point addition, provided that the curve is non-singular. If $char(\mathbb{F}) \neq 2, 3$, this is equivalent to $X^2 + aX + b$ having no repeated factors, or that the discriminant $4a^3 + 27b^2 \neq 0$.

To explain this group law, we consider the case $\mathbb{F} = \mathbb{R}$. The rules for point addition can be summarized as:

> The sum of three points where a line intersects the curve is O.

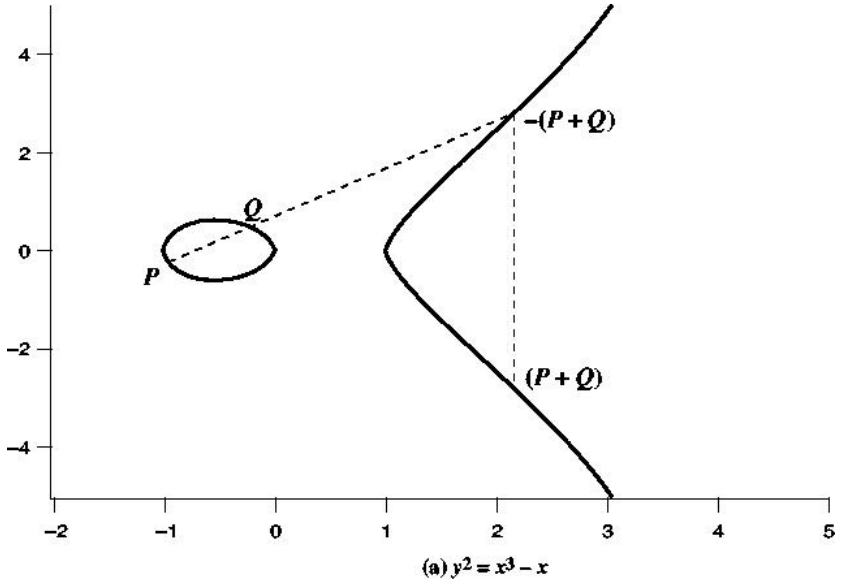
Note. The point at infinity is the "third point of intersection" of any vertical line with the curve.

This idea leads to the following rules for adding and inverting points.

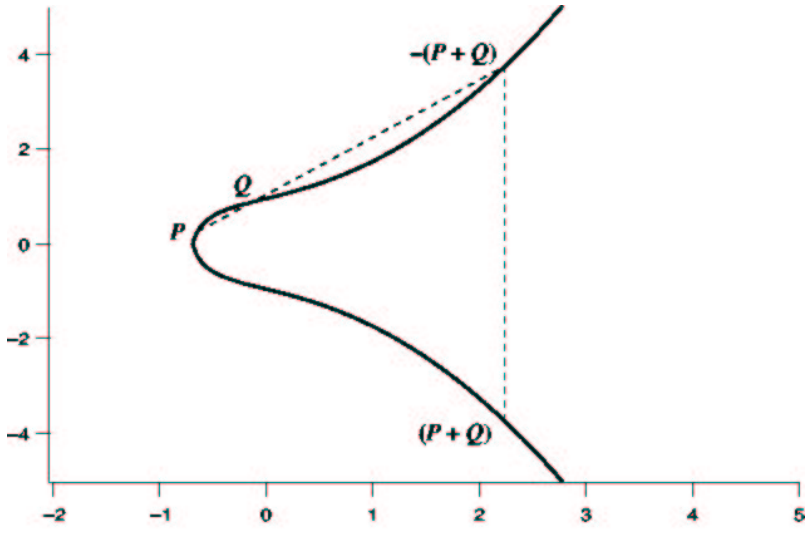
Definition 8.3. Let P, Q be points in $E(\mathbb{R})$ for some elliptic curve E. The negative of P (given by $-P$) and the sum of P and Q (given by $P + Q$) are defined as follows:

1. If $P = O$, then $-P = O$. For any point Q we define $O + Q = Q$ — O serves as the additive identity. In what follows, assume neither P nor Q is O.
2. If $P = (x, y)$, then $-P = (x, -y)$ (reflection across the x-axis). We define $P + (-P) = O$.
3. If P and Q have different x-coordinates, then a line through P and Q intersects E in exactly one more point R, unless the line is tangent to P in which case we take $R = P$, or tangent to Q in which case we take $R = Q$. We define $P + Q = -R$.
4. If $P = Q$, then the line tangent to P intersects E in exactly one other point R. We define $P + P = 2P = -R$.

The following graphs illustrate two typical elliptic curves over $\mathbb{R}$ and the general case of point addition.



(a) $y^2 = x^3 - x$



(b) $y^2 = x^3 + x + 1$

Formulas for point addition can be found by computing the third intersection point R.

- The line through P and Q has equation $y = \lambda x + \beta$, where $\lambda = (y_2 - y_1)/(x_2 - x_1)$ and $\beta = y_1 - \lambda x_1$.
- To find intersection points, substitute $y = \lambda x + \beta$ into the equation for E and solve for x.
- The x coordinates of the three intersection points, x_1, x_2, and x_3 are the roots of $x^3 - (\lambda x + \beta)^2 + ax + b$.
- $x_1 + x_2 + x_3 = \lambda^2$ (property of roots of monic polys) so $x_3 = \lambda^2 - x_1 - x_2$

Thus, for $char(\mathbb{F}) \neq 2, 3$ and $E : Y^2 = X^3 + aX + b$, we obtain $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1, \quad \lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q. \end{cases}$$

Example 8.1. $P = (-3, 9)$ and $Q = (-2, 8)$ are points on $Y^2 = X^3 - 36X$. $P + Q = (6, 0)$ and $2P = (25/4, -35/8)$ (both are also on the curve).

Definition 8.4. Let P be a point on $E(\mathbb{F})$ and $n \in \mathbb{N}$. Define $nP = P + P + \ldots P$ (P added to itself n times).

8.6.2 Elliptic Curves in Cryptography

Consider $E(\mathbb{F}_q)$ where $\mathbb{F}_q$ is the finite field of q elements. Then $|E(\mathbb{F}_q)|$ is finite, as there are only q possible values for each point coordinate. A theorem of Hasse states

$$q + 1 - 2\sqrt{q} \leq |E(\mathbb{F}_q)| \leq q + 1 + 2\sqrt{q}.$$

i.e., $|E(\mathbb{F}_q)|$ is roughly as large as q.

The geometric analogue of point addition does not carry over to the finite field case, but the algebraic formulas still work. Thus, $E(\mathbb{F}_q)$ is a finite abelian group under point addition.

Elliptic curves over $\mathbb{F}_p$ where p is a large prime admit efficient software implementations. The formulas are the same as above if $char(\mathbb{F}_p) > 3$.

Example 8.2. Let $E : Y^2 = X^3 + X + 1$. Then $P = (3, 10)$ and $Q = (9, 7)$ are both points in $E(\mathbb{F}_{23})$. We have $P + Q = (17, 20)$ and $2P = (7, 12)$.

Elliptic curves over $GF(2^n)$ are attractive because they admit efficient hardware implementations. If $E : Y^2 + XY = X^3 + aX^2 + b$ (the non-supersingular case) the formulas for addition are

$$x_3 = \lambda^2 + \lambda + a + x_1 + x_2, \quad y_3 = \lambda(x_1 - x_3) + x_3 + y_1, \quad \lambda = \left(\frac{y_2 + y_1}{x_2 + x_1}\right)$$

and the doubling formulas are

$$x_3 = \lambda^2 + \lambda + a, \quad y_3 = x_1^2 + (\lambda + 1)x_3, \quad \lambda = x_1 + \frac{y_1}{x_1} \ .$$

Since $E(\mathbb{F}_q)$ is a finite abelian group under point addition, it can be used in any generic protocol like Diffie-Hellman or El Gamal. The additive variant of g^x is computing xP, which can also be done efficiently with the binary exponentiation algorithm. The corresponding discrete logarithm problem is to compute x given points P and xP.

There is a polynomial-time algorithm for computing $|E(\mathbb{F}_q)|$ due to Schoof (Math Comp 1985). In practice, $|E(\mathbb{F}_q)|$ can be computed for q up to several hundred digits, allowing elliptic curves to be used easily in signature schemes.

Except for a few special cases, the best-known algorithms for solving the elliptic curve discrete logarithm problem are exponential in $\lg q$, namely $O(\sqrt{q})$. Some special cases:

- $|E(\mathbb{F}_q)| \mid q^i - 1$ for "small" i — Weil and Tate pairing reduce the ECDLP to DLP in $\mathbb{F}_{q^i}$.
- $|E(\mathbb{F}_q)| = q$ — *anomolous* curve, ECDLP reduces to DLP in $\mathbb{F}_q$.
- $\mathbb{F}_{2^{nm}}$ — Weil descent may reduce ECDLP to DLP on a hyperelliptic curve (which under certain conditions is subexponential)

Week 9

Key Management and Authentication

We have seen that symmetric cryptography is well-suited for providing

- confidentiality (bulk encryption)
- data integrity (hash function, MAC)

and public-key cryptography for

- data origin authentication (digital signatures)
- shared secret key agreement (key exchange)

Most cryptographic systems are hybrid systems which use both symmetric and asymmetric techniques to provide the above services.

We have seen symmetric and asymmetric primitives which accomplish all four of these tasks and are secure in the presence of *passive* adversaries. Problems arise when adversaries are assumed to be *active*.

Example 9.1. Consider the following simple PKC-based key exchange protocol:

1. A randomly selects a symmetric key K.
2. A encrypts K using B's public key and sends $E_B(K)$ to B.
3. B computes $K = D_B(E_B(K))$.

If B uses any of the public-key cryptosystems we have seen (eg. RSA, El Gamal) then this is secure against passive attacks. However, C can *impersonate* B by substituting his public key for B's, thereby acquiring K.

Example 9.2. Impersonation can also be used to forge digital signatures. If C convinces B that his public key belongs to A, then B will attempt to verify signatures from A using C's public key.

Example 9.3. A similar *man-in-the-middle* attack can be mounted against Diffie-Hellman if an active adversary substitutes his own g^x and g^y values for those sent by A and B.

To address these problems, it is necessary for A and B to be able to verify the *authenticity* of their respective public keys.

9.1 Public-Key Infrastructures

A public-key infrastructure (PKI) consists of a set of techniques and procedures supporting key management for public-key cryptography. PKI supports:

- initialization of system users,
- generation, distribution/authentication, and installation of public and private keys,
- controlling the use of keys (eg. life cycles of session keys, public and private keys),
- update, revocation, and destruction of keys (eg. managing compromise of private keys),
- storage, backup/recovery, and archival of keys (eg. maintaining an audit trail).

We will focus on mechanisms for distribution and authentication of public keys. The vast majority of key distribution systems involve a trusted third party, although we will see see one example (PGP secure email) that uses peer authentication.

Some possible methods for public key authentication:

1. Point-to-point delivery over a trusted channel such as personal exchange, registered mail, courier, etc. Problems: slow, inconvenient. potentially expensive.
2. Direct access to a trusted public file (public-key repository). Advantage: no user interaction. Problems:
   - The repository must be secure and tamper-proof (otherwise impersonation is still possible),
   - Users must have a secure channel (see Point 1) to initially register their public keys.
3. An on-line trusted server dispenses user's public keys on request. The server signs the transmitted keys with its private key. Problems:
   - all users must know the server's public verification key,
   - the trusted server must be online and may become a bottleneck,
   - a communication link must be established with both the server and the intended recipient
   - the server's public-key database may still be subject to tampering.
4. Off-line server and certificates (certification authorities),
5. Use of systems implicitly guaranteeing authenticity of public parameters (identity-based systems).

9.1.1 Certification Authorities

Definition 9.1. A public-key *certificate* is a data structure consisting of a data part (containing at least user identification and the corresponding public key) and a signature part consisting of the digital signature of a certification authority over the data part.

A certificate may also include information such as:

- A time-stamp indicating the currency of the certificate (to facilitate key changing and revocation),
- Additional information about the key (algorithm, intended use),
- Key status (for revocation),
- Signature verification information (CA's name, signature algorithm used)

Definition 9.2. A *certification authority* (CA) is a trusted third party whose signature on a certificate vouches for the authenticity of the public key bound to the subject entity.

Idea: CA issues public-key certificates that may be verified off-line, i.e., users may exchange authentic public keys without having to contact the CA.

B uses a public-key certificate to obtain the authentic public key of A as follows:

1. (One-time) acquire the authentic public key of the CA.
2. Acquire a public-key certificate corresponding to A. This may be over an insecure channel such as a central database, or even directly from A.
3. Verify the authenticity of the public key:
   (a) Verify the currency of the certificate using the time-stamp,
   (b) Verify the signature on A's certificate using CA's public key,
   (c) Verify that the certificate has not been revoked.
4. If all the checks succeed, accept the public key in the certificate as A's public key.

Requirements of the scheme:

1. Any participant can read a certificate to determine the name and public key,
2. Any participant can verify that the certificate originated from the CA and is not counterfeit,
3. Only the CA can create and update certificates,
4. Any participant can verify the currency of the certificate.

Users must register their public keys with the CA in a secure manner (typically in person). The CA's public key (required for certificate verification) may be obtained at that time. If the user generates his own keys, the CA must verify the source of the keys. In all cases, the CA must verify the binding between the public and private keys.

Users whose private keys are compromised may have the CA revoke the old certificate (via the time stamp or other fields) and issue a new certificate containing a new public key. This is analogous to canceling a credit card.

Note. X.509 is a standard for such a certificate-based authentication scheme. VeriSign is one example of an online CA service that uses X.509 certificates. Users request certificates for their public keys via the VeriSign website. Certificates are issued with three authentication levels:

- Class 1 — User's email address is confirmed by sending a PIN and instructions to pick up the certificate.
- Class 2 — Class 1, plus user's application information is compared with an online consumer database. Confirmation is sent to the user-specified postal address, indicating that a certificate has been issued.
- Class 3 — User must prove his identity by providing notarized credentials or applying in person.

9.1.2 Identity-Based Cryptography

Idea: bind public keys directly to a user's identity

Definition 9.3. An *identity-based cryptosystem* (ID-based) is a PKC in which an entity's public identification information (unique name) plays the role of its public key. The unique name is used by a trusted authority T to compute the entity's corresponding private key.

The public key is typically constructed from the unique name using some publicly known function.

Motivation — ideal mail system. Knowledge of a person's name alone is sufficient to

- send mail which can be read by that person only (secure),
- allow verification of signatures that person alone could have produced.

The binding of a unique entity name to the public key eliminates the need need for public key authentication. If the wrong public user data is used, the cryptographic transformations fail. Advantages:

- users need not exchange keys,
- public directories (files of public keys or certificates) need not be kept,
- the trusted authority is only required during the set-up phase (to compute private keys)

Key revocation may be addressed by incorporating a key validity period into the ID string used to compute a user's public key.

Feige, Fiat, and Shamir (Journal of Cryptology 1998) proposed an ID-based signature scheme based on computing square roots modulo pq (p and q large primes).

Boneh and Franklin (CRYPTO 2001) proposed the first practical ID-based *encryption* scheme using the Weil pairing on elliptic curves.

Week 10

Cryptography in Practice

10.1 Email Security and PGP

PGP — Pretty Good Privacy, secure email system developed by Phil Zimmerman. Features:

- Free world-wide availability, runs on most platforms,
- Use best available cryptographic primitives,
- Not developed by government nor standards organization,
- Compatible with most email programs,
- Automatically segments large messages (to accommodate message size limitations).
- Users may have multiple public keys. Each key is identified by its 64 low-order bits (key ID, denoted ID_K).

PGP provides the following cryptographic services:

- Authentication (using digital signatures). DSS/SHA (1024 bit keys) and RSA/SHA (768 to 3072 bit keys) are supported.
- Confidentiality (using 64-bit CFB symmetric encryption). CAST-128, IDEA (128-bit), and 3DES (168-bit) are supported for encryption, El Gamal and RSA for key exchange.

A sends a message M (authenticated and encrypted):

1. Compute signature $S = D_A(H(M))$ on the SHA-1 hash value of M
2. Compress (S, M) (using ZIP)
3. Generate a random session key K to be used to encrypt *only* this message,
4. Encrypt a time-stamp T, key ID of A's public key, the signature S, and M, to obtain the ciphertext $C = E_K(T, ID_K(E_A), S, M)$.
5. Encrypt K using B's public key and send $(ID_K(E_B), E_B(K), C)$

B decrypts the session key $K = D_B(E_B(K))$ recovers the signature S and message M using $D_K(C) = (T, ID_K(E_A), S, M)$, and verifies the authenticity of the message by comparing $H(M)$ with $E_A(S)$.

Some features:

- The key IDs allow A and B to use the correct public keys when they have multiple public/private key pairs.
- The use of time-stamps prevents replay attacks (an adversary resends an intercepted valid signature at a later time).
- Session keys are not needed — each symmetric key is used to encrypt only one message.

10.1.1 Key Management in PGP

Three types of keys:

- Session keys (one-time message encryption),
- Public/private keys (user's public/private key pairs, other users' known public keys)
- Pass phrase (for encrypting key ring files)

Random and Pseudorandom Numbers

Session keys are generated pseudorandomly on demand as follows (based on ANSI X9.17 block cipher based pseudorandom number generation):

- Compute random 128-bit key K and random 64-bit IV X_0,
- Compute pseudorandom 64-bit blocks X_i $(i = 1, 2, \ldots)$ as $E_K(X_{i-1})$.
- Use two pseudorandom blocks X_i, X_{i+1} as the 128-bit block cipher key.

A 256-byte buffer of true random data is used to seed the pseudorandom number generator and to generate public key pairs. The random data is generated by measuring the latency between key strokes and their content.

Key Rings and Authentication

A user's public/private key pairs are stored in a *private key ring*. Each entry (corresponding to one public/private key pair) contains:

- time-stamp,
- key ID,
- public key (generated by PGP),
- private key (generated by PGP),
- user ID (different ID's, typically email addresses, may be assigned to different public/private keys)

The private key ring file is maintained and stored by the user. The private keys are stored in encrypted form (block cipher). The access key is the SHA-1 hash value of a secret pass phrase.

Others' public keys known to a user are stored in a *public key ring* (maintained and stored by the user). An entry (corresponding to one known public key) contains:

- time-stamp,
- key ID,
- public key,
- owner trust field — is this public key trusted to sign other certificates? User-assigned — higher value indicates higher degree of trust.
- user ID (email address),
- key legitimacy — higher value indicates higher trust in the binding of public key to user ID. Computed by PGP as a function of signature trust fields,
- digital signatures — zero or more signatures, each vouches for the authenticity of the key to ID binding of this public key,
- signature trust field — indicates the degree of trust in one signature. Higher value indicates a higher degree of trust in the signature's author. The key legitimacy field is a function of the signature trust fields.

Other users' authentic public keys can be obtained using secure public channels, CA's, etc..., *plus* from a mutually trusted individual. PGP provides a mechanism for *quantifying* trust.

A inserts a new public key (certificate with attached signatures) into his public key ring as follows:

- The owner trust field is assigned (byte value).
  - If the public key belongs to A, a value of "ultimate trust" is assigned.
  - Otherwise, user selects one of "unknown," "untrusted," "marginal trust," or "complete trust."
- Signature trust fields are assigned (byte values). For each attached signature:
  - If the signature's author is unknown, the signature trust field is assigned "unknown user."
  - Otherwise, the signature trust field is assigned the corresponding owner trust field.
- The key legitimacy field is evaluated based on the signature trust fields.
  - If at least one signature trust field is "ultimate trust," key legitimacy is assigned "complete."
  - Otherwise, key legitimacy is a weighted sum — $1/X$ if signature trust field is always trusted, $1/Y$ if usually trusted (X and Y user-configurable parameters).
  - If total weight is ≥ 1, key legitimacy is assigned "complete" (at least X always trusted signatures or Y usually trusted). Otherwise, key legitimacy is "not trusted" or "marginally trusted."

This scheme makes it possible to trust the authenticity of a user's public key, but not to trust the user to sign other's keys.

A user can revoke his public keys by issuing a signed certificate (with a revocation flag set) and sending it to as many users as possible.

10.1.2 S/MIME

IETF standardization track. Similar to PGP in functionality and cryptographic protocols supported. Only CA's can create certificates (using X.509 version 3), and users must register their public keys.

10.2 Web Security

Challenges to web security:

- *very* widely used, often as highly-visible outlets for information (motivation to break)
- two-way (interactive) traffic (servers can be attacked)
- by subverting a web server it may be possible to access an organization's entire computer network
- web software is very complex but easy to use — low level security risks are often buried
- anyone (regardless of security training) can use the web

Threats to web security:

- Confidentiality
  - eavesdropping
  - theft of information/data
  - information about client identities
- Integrity
  - modification of data, memory, or messages
- Authentication
  - impersonating legitimate users
  - data forgery
- Denial of service
  - flooding machine with bogus threats
  - filling up disk or memory

Approaches to web security:

1. Network level (IPsec)
   - general-purpose secure modification of the low-level internet protocol (IP)
   - transparent to applications and end-users
2. Transport level (SSL or TLS)
   - can be transparent to applications or built-in (eg. Netscape and Internet Explorer)
   - SSL (secure socket layer) widely used, also TLS (transport layer security)
3. Application level (Kerberos, PGP, SET)
   - security infrastructure tailored to the specific application
   - eg. SET — secure electronic transaction (for electronic commerce)

In all cases, the implementation should allow for easy interchange of the cryptographic primitives (in case one is broken).

10.2.1 Secure Socket Layer (SSL)

Developed by Netscape, became IETF standard TLS (transport layer security). Makes use of TCP to provide a reliable end-to-end secure service.

Two layers of protocols:

- SSL Record protocol — provides basic security services (confidentiality and message integrity) to upper-level protocols (eg. HTTP)
- SSL Handshake protocol, SSL Change Cipher Spec protocol, SSL Alert protocol — upper-level protocols used to set up and manage a secure connection

Connection — a peer-to-peer communication channel (transport). The SSL record protocol is used to secure the communications. A connection state is defined by parameters including MAC keys, cipher keys, and IVs (for CBC encryption). Each connection belongs to a session.

Session — association between client and server for which a set of cryptographic parameters are defined. The SSL Handshake protocol is used to create a session (negotiate cryptographic parameters, authenticate parties). A session state is defined by parameters including the cipher to be used for connections and an X.509.v3 certificate of its peer.

SSL Record Protocol

Application data is sent via a connection as follows:

1. The data is broken into 16384-byte blocks.
2. Each block is compressed.
3. A MAC is computed for each block and appended.
4. The block plus MAC is encrypted.
5. An SSL record header is attached and the block is sent to the peer.

The MAC is performed using a combination of a shared secret key and SHA-1 or MD5 (TLS uses HMAC).

IDEA-128, RC2-40, DES-40, DES, 3DES, and Fortezza are supported for block cipher encryption.

RC4-40 and RC4-128 are supported for stream cipher encryption.

SSL Handshake Protocol

Used to initialize a session:

- Client and server authenticate each other (via X.509.v3 certificates)
- Encryption, MAC, and required keys are negotiated

No application data is sent until the handshake is successfully executed.

Phase 1 — Establish Security Capabilities

- Client sends a "client-hello" message to the server containing:
  - highest SSL version understood by the client

  - time stamp and 28 random bytes (used as nonces to prevent replay attacks
  - CipherSuite — cryptographic algorithms supported by the client listed in decreasing order of preference. Each entry contains a key exchange algorithm and a CipherSpec (encryption and MAC specifications)

- Server responds with a "server-hello" message containing the same information.
  - SSL version is the highest mutually-supported version
  - CipherSuite contains the single cipher suite selected from the client's list

Key exchange algorithms supported:

- RSA,
- Fixed Diffie-Hellman — g^x contained in a certificate,
- Ephemeral Diffie-Hellman — g^x randomly generated and signed by each peer. For one-time secret keys.
- Base Diffie-Hellman (no authentication),
- Fortezza

Phase 2 — Server Authentication and Key Exchange

- Server sends a "certificate-message" containing its certificate (if authentication is required)
- Server sends a "server-key-exchange" message to initiate key exchange. Not required if RSA, Fixed Diffie-Hellman.
- Server sends a "certificate-request-message" if client authentication is required. This message specifies the type of signature algorithm to be used and a list of acceptable certification authorities.
- Server sends a "server-done" message and wait for the client's response.

RSA, DSS, and Fortezza are supported for signatures.

Phase 3 — Client Authentication and Key Exchange
If the server's certificate is valid and the "server-hello" message is valid, the client sends the following messages (if necessary):

- A "certificate-message" containing its certificate or a "no-certificate" alert.
- A "client-key-exchange" message containing the clients response for the key exchange protocol.
- A "certificate-verify" message containing a signature on the client key exchange message (if required by the key exchange protocol)

Phase 4 — Finish

- Client sends a "change-cipher-spec" message and activates the algorithms in the CipherSpec
- Client sends a "finished" message encrypted with the new CipherSpec
- Server sends a "change-cipher-spec" message and activates the same CipherSpec
- Server sends a "finished" message encrypted with the new CipherSpec

After this protocol, all the information required to secure connections has been exchanged and application data may be transferred.