⇒ Stream Ciphers – Week1 (Nov 6, 2012)

A cipher: is a pair (E, D) over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where
($\mathcal{K}$: Set of all possible keys (KEYSPACE); $\mathcal{M}$: set of all possible messages; $\mathcal{C}$: Set of all possible cipher-text)

$E: \mathcal{K} \times \mathcal{M} \to \mathcal{C}$

$D: \mathcal{K} \times \mathcal{C} \to \mathcal{M}$

$\forall m \in \mathcal{M}, k \in \mathcal{K}\ \ D(K, E(K, m)) = m$ ← consistency requirement

(all "useful" ciphers must satisfy)

E & D are efficient algorithms. E is randomized often and D is always deterministic.

One time pad (OTP): $\mathcal{M} = \mathcal{C} = \{0,1\}^n = \mathcal{K}$ $\quad E = D = \oplus$ (xor)

Vernam – 1917 – is a very secure cipher – cannot be broken by a ciphertext only attack.. Not practical because the key length is too long

Shannon 1949 → defined what it means to be a "good cipher"

A good/secure cipher is one where looking at the cipher text does not give you any information about the plaintext. Shannon formalized this intuition as:

Cipher (E, D) over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ has "perfect secrecy" if:–

$\forall m_0, m_1 \in \mathcal{M}$ where $len(m_0) = len(m_1)$ and $c \in \mathcal{C}$

$$Pr[E(K, m_1) = c] = Pr[E(K, m_2) = c] \quad \text{where } K \xleftarrow{R} \mathcal{K}$$

(random variable uniformly sampled over $\mathcal{K}$)

OTP has perfect secrecy:

$$Pr[E(K, m_1) = c] = \frac{\#\text{keys } K \in \mathcal{K} \text{ such that } E(K, m) = c}{|\mathcal{K}|}$$

(if (not a variable) ↑ $|\mathcal{K}|$ ← not a variable)

if the no. of keys that generates a given cipher text from a given plaintext is constant, then the Pr will be fixed and equal.

For OTK this no. is $\frac{1}{|\mathcal{K}|}$.

Shannon also proved

Perfect secrecy requires $|\mathcal{K}| \geq |\mathcal{M}|$

for OTP $|\mathcal{K}| = |\mathcal{M}|$ ↓ optimal perfect secrecy cipher

STREAM CIPHERS & PSEUDO RANDOM GENERATORS (Nov 6, 2012)

A stream-cipher is a practical implementation of OTP. Instead of using a random key we use a pseudo-random key using a PR Generator.

PRG $\Rightarrow$ $G: \{0,1\}^s \to \{0,1\}^n$ where $n \gg s$

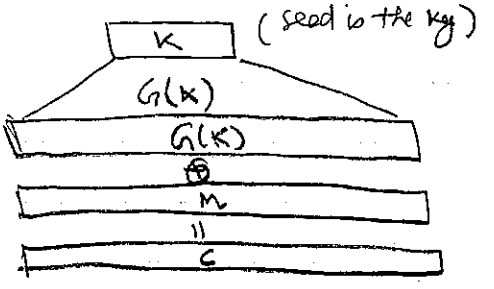
deterministic algorithm that takes a "seed" of length s and generates a bit stream of length 'n'

G must be efficient + deterministic (source of randomness is only the seed)

$C: (E, D)$ where

$E := E(k, m) = m \oplus G(k)$

$D := D(k, c) = c \oplus G(k)$



note k length < m length so there is no perfect secrecy.

Quality of PRG is important: be unpredictable. A PRG is predictable if must not given the first i bit of its output we can compute its $i+1^{th}$ bits.

$G: K \to \{0,1\}^n$ is predictable if

$\exists$ "efficient" algorithm A and $\exists i$ $1 \le i \le n-1$ such that

$$\underset{k \xleftarrow{R} K}{Pr}\left[A(G(k)|_{1 \dots i}) = G(k)|_{i+1} \right] \ge 1/2 + \epsilon$$

where ϵ is non-negligable: practical $\epsilon \le 1/2^{80}$

PRG is unpredictable if $\forall i$: no efficient can predict $i+1^{th}$ bit with ϵ non-negligible

Linear congruential Generator (LCG) weak PRG | random glibc based on LCG) DO NOT USE.

Theoretically $\epsilon : Z^{\ge 0} \to R^{\ge 0}$

ϵ is non negligible is $\exists d: \epsilon(x) \ge 1/x^d$ for infinitely many values of x chosen

i.e. it is a function that will take infinitely many values greater than the inverse of a d^{th} degree polynomial

ONE TIME PAD ATTACKS

Nov 7 2012

ONE MESSAGE ⇒ ONE KEY

Stream ciphers are approximations of the OTP by replacing the impractically long keys with a PRG (pseudo random generator) and a short key (seed) (3)

$G: K \to \{0,1\}^n$ $\quad E := E(k,m) = m \oplus G(k)$, $\quad D := D(k,c) = c \oplus G(k)$

Attack 1 : One time pads must be used only "once"! Two messages xored with same pad (or its approximation, same output of G). can be xored to get an xor of plaint

— MS-PPTP (point to point Transfer Protocol) : Client and server use the same OTP which means requests & responses are encrypted with same PR seed. for client-sovery remember to keep separate keys for client & sereves.

a key pair must be used - one for client's usage one for servers.

- 802.11b WEP : bad usage of stream ciphers:

OTP IS MALLEABLE

WANTED to use different key for every packet but did not!
- wanted to use different keys for different messages
- but, the way they differentiated keys was wrong
- keys were changed by prefixing a 24 bit IV to the 104 bit key. The IV served as a counter. started at 0, incremented with each packet.
- power reset re-initializes IV to 0. It anyway repeats after 2^{24} pkts

WEP - had other problems. The different keys are similar (they have the same 104 bits at the end). for the PRG used in WEP RC4. For this PRG, with the keys used (related) if we capture 1M packets, the key can be derived.

WEP is broken because: it uses 2 time pad + its key can be derived if sufficient packets are captured.

A better design would have used the long term key to generate a PRG value and the output of the PRG should have been used as the seeds for different generation keys for different keys)

DISK ENCRYPTION - with stream ciphers is a bad idea because when files are edited we can effectively get multiple blocks encrypted with the same OTP. Effectively different edit/save will lead to 2TP.

ANOTHER PROBLEM WITH OTP - OTP IS MALLEABLE (does not guarantee integrity)

$C = m \oplus K$

$c \oplus p$ - will lead to a different message to be decrypted

↑ perturbation

modifications to ciphertext are undetectable and can be tweaked to modify the meaning of the message

RC4: Quite popular, takes 128 bit seed (variable length also supported) and expands it to 2048 bits. Then these bits are looped over to generate bytes of random data. One byte per round. Has several weaknesses and is not recommended:

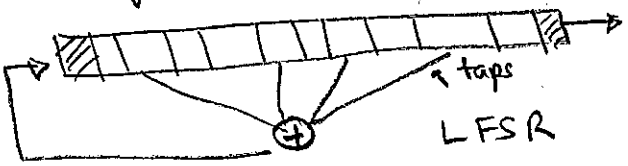
→ Bias in initial output: more likely to be 0^s than 1. (work around is to ignore first 256 bytes)

→ After a while (0,0) are more likely than they should be

- If used with related keys - then it is possible to recover the root key (As is done in WEP)

LFSR|CSS (Content scrambling system) – Linear Feedback Shift Register – Hardware implementation of a PRG

Seed is the initial bit sequence in the Register. At every round the register is shifted right one bit falls off, an XOR of a few bits is fed to the left.



– DVD Encryption, GSM, Bluetooth

hardware encryption prefers LFSR implementations

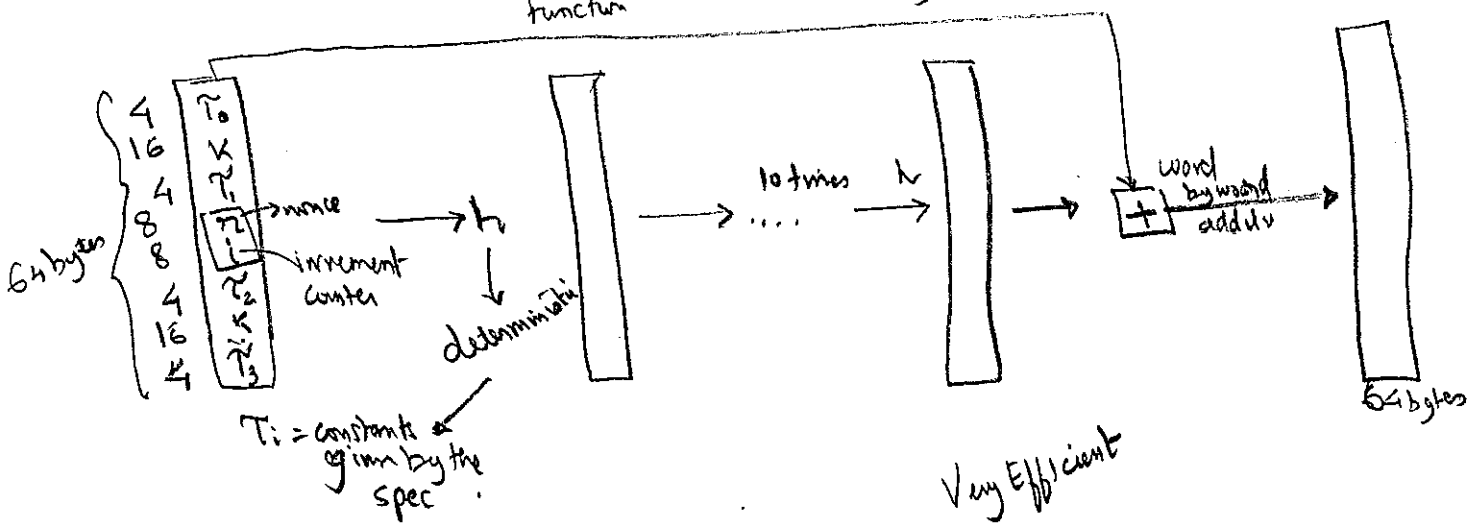
eStream Project – Qualified 5 stream ciphers. PRG: $\{0,1\}^s \times R \to \{0,1\}^n$ (R: nonce)

Use a nonce along with the key. $\{k, r\}$ is made unique and the same key can be used by changing the nonce.

Salsa 20: takes either 128, 256 bit key, 64 bit nonce

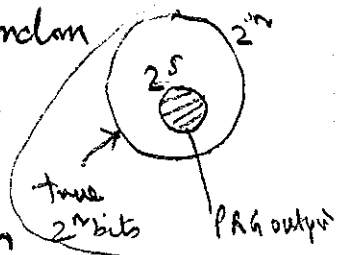
$$\text{Salsa20}(k; r) := H\big(\underbrace{k}_{\text{key}}, (\underbrace{r}_{\text{nonce}}, 0)\big) \,||\, H(k,(r,1)) \ldots\ldots$$

(H: function; 0: incrementing counter)



PRG SECURITY DEFINITIONS

What does it mean to say a "good PRG". A good PRG is one which outputs bits that "look" random. It cannot be truly random because we have only s random bits and were outputting n bits. A true uniform random distribution of over $\{0,1\}^n$ can emit anything inside the big circle but a PRG has only enough info to generate a small sample of it. So at best its output can look random to an adversary. More formally, to an external observer given an output from the PRG and one from a true random distribution should look same. This "looks the same" is formalized as the notion of ADVANTAGE



$$Adv_{PRG}[A,G] := \left| \Pr_{K \to \mathcal{K}}[A(G(K))=1] - \Pr_{r \xleftarrow{R} \{0,1\}^n}[A(r)=1] \right|$$

Imagine an efficient procedure A, that outputs 1 when it believes that the bit string it is passed is random. We call A a "statistical test" because the only way to implement it would be to rely on certain patterns once expects statistically present in true random bits.

$G: K \to \{0,1\}^n$

$Adv_{PRG}[A,G]$ — Advantage of a statistical test 'A' over PRG 'G'

$\Pr_{K \to \mathcal{K}}[A(G(K))=1] \sim X$ — the probability that A declares an output of G as random

$\Pr_{r \xleftarrow{R} \{0,1\}^n}[A(r)=1] \sim Y$ — The probability that A declares a true random string as random.

$Adv_{PRG}[A,G] = |X - Y|$ i.e advantage is the difference between the 2 probabilities. Intuitively it measures how often the test A distinguishes between the 2.

$1 \Rightarrow$ totally breaks PRG
$0 \Rightarrow$ does not distinguish true random from PRG.

$\{K \xleftarrow{R} \mathcal{K} : G(K)\} \approx_p^{uniform(\{0,1\}^n)}$

$Adv_{PRG}[A,G] = [0,1]$

$G: K \to \{1,0\}^n$ is a secure PRG if $\forall$ efficient statistical tests A: $Adv_{PRG}[A,G]$ is negligible

Provably secure PRG don't exist — there are heuristic candidates like SalSa20

All secure PRG are unpredictable

If PRG is predictable then we can define a statistical test with

YAO proved if next bit prediction cannot distinguish G from random, then G is secure

SEMANTIC SECURITY

Practical stream ciphers are not "perfectly secure" (as Shannon defined them) so we need a weaker measure for them. That's <u>semantic security</u>.

We begin with the notion of "computationally indistinguishable" ($\approx_p$) — polynomial time

Two distribution P_1 & P_2 are computationally indistinguishable ($P_1 \approx_p P_2$) if for all efficient statistical tests A:

$$\left| \Pr_{x \leftarrow P_1}[A(x)=1] - \Pr_{x \leftarrow P_2}[A(x)=1] \right| < \text{negligible}$$

In this notation secure PRG can be defined as

$$\left\{ k \xleftarrow{R} K : G(k) \right\} \approx_p \text{uniform}(\{0,1\}^n)$$

Stream ciphers are semantically secure if they use a secure PRG

Shannon's perfect secrecy in words the distribution of the encryption of any two messages is the <u>same</u>

E, D has "perfect secrecy" if $\forall m_0, m_1 \in M \quad |m_0| = |m_1|$

$$\{E(K, m_0)\} = \{E(K, m_1)\} \quad K \leftarrow \mathcal{K}$$

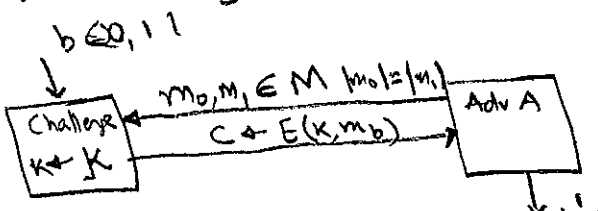
(requires keys as long as messages)

$\Downarrow$ weakened

Semantic Security $\Leftarrow$ $\{E(K, m_0)\} \approx_p \{E(K, m_1)\}$ and require the adversary to actually produce m_0, m_1

the adversary outputs two messages m_0, m_1 and the challenger outputs two ciphertexts c_1 & c_2 without telling the adversary which one is the encryption of which (of the same length)

The adversary takes c_1 & c_2 and outputs 0 & 1 based on which of m_0 & m_1 it thinks the encryption belongs to. If it can do this quite correctly then the cipher is not 'semantically secure'.



<u>Intuitively</u> if the adversary can produce 2 messages such that it can tell the encryption of 1 from the other – he has broken the cipher.

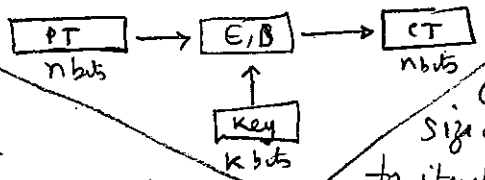
formally $b \in \{0,1\}$ W_b = [event that output of adversary $E^K(b) = 1$]

$$Adv_{ss}[A, E] = \left| \Pr[W_0] - \Pr[W_1] \right| \in$$

A symmetric cipher E is semantically secure if for all efficient adversary A $Adv_{ss}[A, E] < \text{negligible}$

If <u>any bit of the PT</u> is leaked in the cipher with a non-negligible probability (any predicate of the PT) then we can choose m_0 & m_1 in a way that for one the predicate is true & the other it is false. Then check for the predicate on the ciphers and since the encryption leaks the predicate truth value we can distinguish m_0 & m_1.

Crypto workhorse. Maps n bits of inputs to n bits of outputs.



Block ciphers are significantly slower than Stream ciphers (more versatile)

Typically use iterations called "rounds". The key passed is expanded to a larger bit size and is split into "round keys". These round keys are used to iteratively encrypt the PT. 3DES - 48 rounds, AES - 10 rounds.

3DES: n = 64 bits, K = 168 bits
AES: n = 128 bits, K = 128, 192, 256 bits

$R(K, m)$ = round function

$K \to K_1, K_2, K_3 \ldots K_n$ (key expansion)

$m_1 = R(K_1, m) \to m_2 = R(K_2, m_1)$

$\ldots \quad c = R(K_n, m_n)$

ABSTRACTIONS FOR REASONING OVER BLOCK CIPHERS

- PRF — Pseudo Random Functions
- PRP — Pseudo Random Permutations

PRP are good mathematical models of block ciphers

$E(K, \cdot)$ notation to fix one argument and let other float

PRF: defined over (K, X, Y) — Keyspace, input set, output set

PRF: $F: K \times X \to Y$

Assume efficient algorithm to compute; not necessarily invertible

PRP: $E: K \times X \to X$

- efficient deterministic algo to compute
- $E(K, \cdot)$ is one-to-one
- efficient inversion algo D exist $D(k, y)$

PRP is a PRF with $X = Y$ & is efficiently invertible

SECURE PRF

Let $F: K \times X \to Y$ be a PRF

also consider $Funs[X, Y]$: set of all possible functions from X to Y. Huge set with $|Y|^{|X|}$ elements. Note F is not an element of $Funs[X, Y]$ because it also takes 'K'. If we bind K by $F(K, \cdot)$ we get $|K|$ functions belonging to Funs

$$S_F = \{ F(K, \cdot) \text{ such that } K \in K \}$$

S_F: $|K|$ — $Funs[X, Y]$: $|Y|^{|X|}$

a PRF is secure if a random function from S_F is indistinguishable from a random function in $Funs[X, Y]$. That is an adversary cannot tell if when given a mapping from a completely random function chosen from $Funs[X, Y]$ and one chosen from S_F which is which. $F(K, \cdot)$ is indistinguishable for $f(\cdot)$ if $F: K \times \{0,1\}^n \to \{0,1\}^n$ is a secure PRF then $G: K \to \{0,1\}^{nt}$

Secure PRFs can be used to make a secure PRG: $G(K) = F(K,0) \| F(K,1) \| F \ldots \| F(K,t)$ (counter mode)

DES (Data Encryption Standard)

Two things that define a block cipher: Key Expansion & Round function.

Key expansion - grows the input key to a larger bit sequence, which is broken into several round keys. each round key is used along with a single round.

Round Function - Iterative application (chained) of the plaintext to get the final ciphertext.

DES → Feistel designed Lucifer at IBM. Key length 128 bits, block size 128 bits.
IBM submitted it (a variant) to National Bureau of Standards (NBS), adopted as DES (56 bits - key, Block len - 64 bits)
in 1997 - by exhaustive search DES was broken → led to AES.

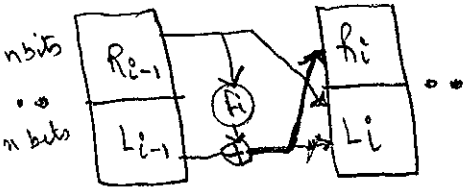
DES Week 2 – #2

DES is a Feistel Network Cipher. A Feistel network is a construction that takes 'd' arbitrary functions and composes an invertible function out of them.

Recollect that all block ciphers are made up of 2 things a "key expansion" routine and a "round function". The "key expansion" takes a key and expands it into a larger bit sequence which is then broken into multiple "round keys". These round keys are fed as input to multiple invocations of the round functions, finally producing the output.

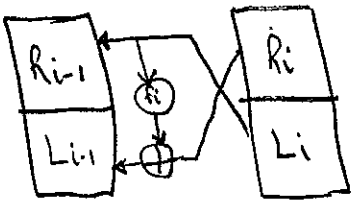
DES started as Lucifer at IBM → which was weakened (bit size reduced to 56) and standardized as DES (in 1976). 56 bits is too small, 1997 it was broken by exhaustive search. Still DES is an amazingly successful cipher.

Given $f_1, \ldots f_d : \{0,1\}^n \to \{0,1\}^n$ (arbitrary) we want to build $F: \{0,1\}^{2n} \to \{0,1\}^{2n}$ where F is invertible.



$R_i = L_{i-1} \oplus f_i(R_{i-1})$ [Forward]
$L_i = R_{i-1}$
$i = 1 \ldots d$

inverse →



(basically start from output and apply functions in reverse order)

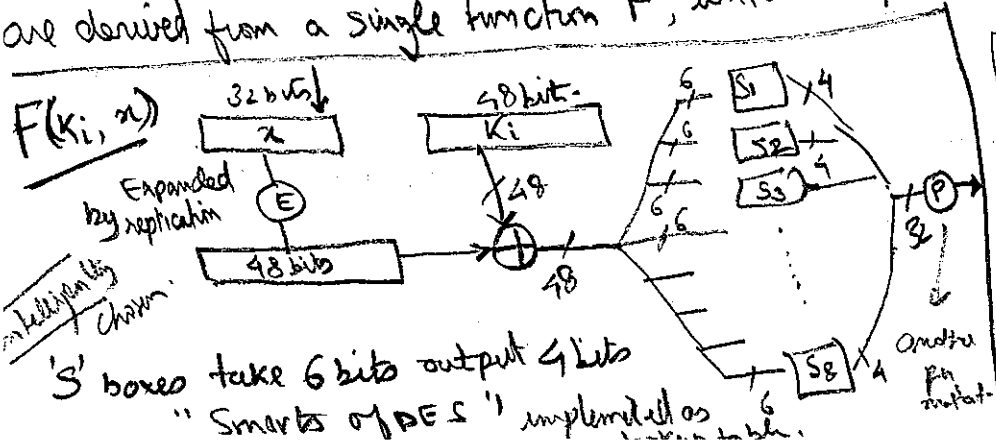
$R_{i-1} = L_i$
$L_{i-1} = R_i \oplus f_i(L_i)$ [inverse]

A Feistel network takes d arbitrary functions and gives us an invertible F.

Luby Rackoff Theorem: If $f: K \times \{0,1\}^n \to \{0,1\}^n$ is a secure PRF then a 3 round Feistel Network $F: K^3 \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ is a secure PRP.

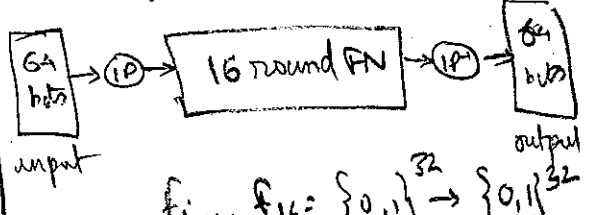
That is, given a secure PRF, if we plug it into a FN and give it 3 independent keys and use it as three round functions in a FN, we get a secure cipher. In another words, if we have a secure PRF Luby Rackoff gives us a secure PRP

DES does exactly that. DES is a 16 round FS. The 16 round functions are derived from a single function F, which is passed 16 different round keys.



'S' boxes take 6 bits output 4 bits

"Secrets of DES" implemented as



$f_1 \ldots f_{16} : \{0,1\}^{32} \to \{0,1\}^{32}$

$f_i(x) = F(K_i, x)$

56 bit key is expanded to 16 48 bit keys

EXHAUSTIVE SEARCH ATTACKS Week2 - #3

Given few input output pairs $(m_i, c_i = E(k, m_i))$ $i = 1 \ldots 3$, find K.

Before we begin, we define "Unicity probability" as the probability that given a message the probability that there is only one key that produced the cipher. In other words, if given a cipher text, we can rest assured that there is hope in trying each of the keys in the key space and be guaranteed that we'll find only one 'key'.

For DES unicity probability is $\left(1 - \frac{1}{256}\right)$ and for AES is $1 - 1/2^{128}$.

Simply: 2^{56} is not too huge and we can break DES by simply trying all possible keys.

First solution was 3DES (Triple DES) $3E : K^3 \times M \to M$, i.e. triple key size to 128 bits

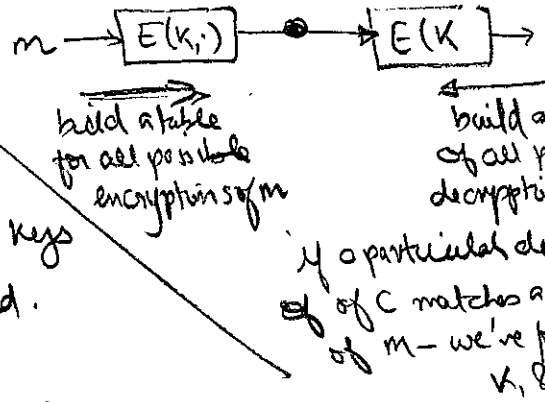
$$3E((K_1, K_2, K_3), m) = E(K_1, D(K_2, (E(K_3, m))))$$

We do EDE and not, EEE because EDE allows us to come down to DES if $K_1 = K_2 = K_3$.

Searching through 128's is hard. But 3DES is 3 times slow.

Quantum computer can break DES in ~~(2^{56})~~ $\sqrt{(2^{56})}$

Why not 2DES 112 is hard too, so why not 2DES. 2DES is susceptible to MITM (Meet in the middle attack)



So → 2DES can be broken in 2^{63} using MITM

Actually 3DES can be broken by MITM in 2^{112} keys but since 2^{112} is good enough, we're not worried.

DESX alternative way to triple the key size $EX : K^3 \times M \to M$

$$DESX((K_1, K_2, K_3), m) = K_1 \oplus E(K_2, K_3 \oplus m)$$

K_1: 64 bits (block size) K_2: 56 bits (DES key) K_3: 64 bits (block size) total = 184 bits

Implementation based Attacks

Measure power usage to predict keys.

LINEAR DIFFERENTIAL ATTACK: If I xor the bits of the message with the cipher text and get some bits of the key, the cipher is broken because then given a large number of (m, c) pairs I can derive the key. Typically m & c have no relation with K, but for DES

$$Pr[m[i_1] \oplus \ldots m[i_n] \oplus c[j_1] \oplus \ldots c[j_v] = K[l_1] \oplus \ldots \oplus K[l_u]] = 1/2 + \epsilon$$

a bug in S5

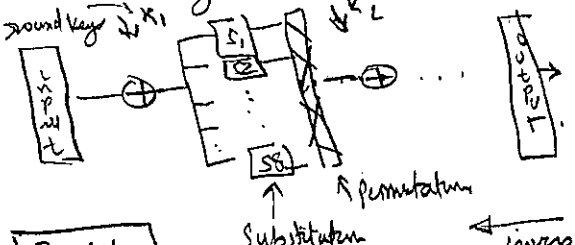
for DES $\epsilon = 1/2^{21}$, so given $1/\epsilon^2$ random m, c pairs, we can do a majority of $m \oplus c$ and get the key bits with a good probability

brings down exhaustive search from 2^{56} to 2^{42}

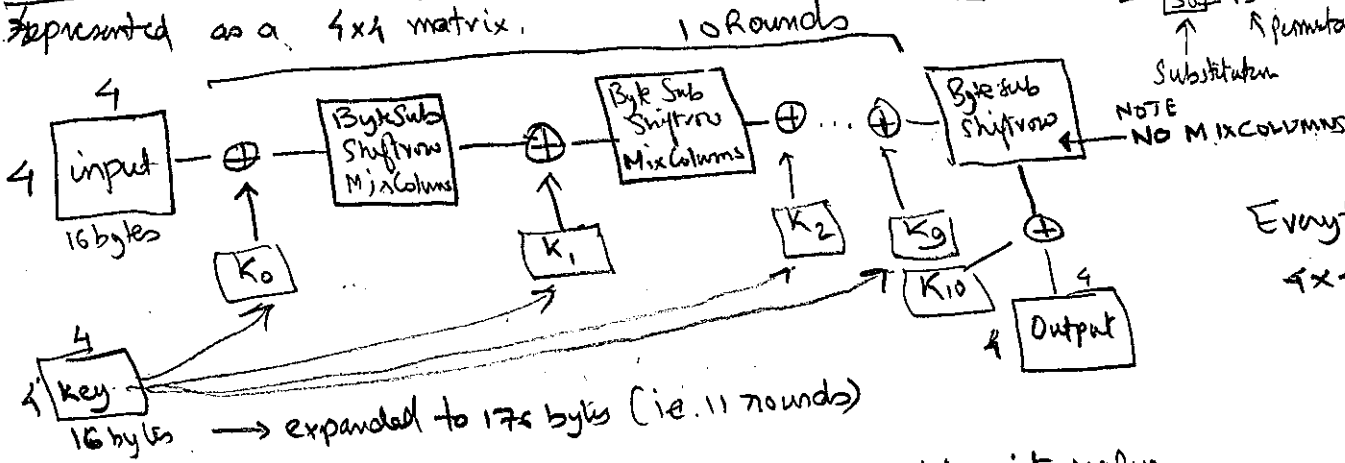
AES

Week 2 # 5

Key sizes 128, 192, 256 bits – Block sizes 128 bits – Not a Feistel cipher. AES is a Substitution-Permutation Network. Chosen from 5 finalists in a NIST selection process in 2000. Unlike DES, every bit is changed in all steps (in DES half the bits are copied as-is).



AES-128. block size of 128 bits, i.e. 16 bytes, i.e. represented as a 4×4 matrix.



Everything is a 4×4 matrix

BYTESUB: a 1 byte S box – Every input cell is replaced by its value taken from the 1 byte Sbox.
Substitution
256 entries one for each of the 256 values that can be present in a given input cell

$$\forall i,j : A[i,j] \leftarrow S[A[i,j]]$$

SHIFT ROWS: Rows in input array are cyclically shifted. 0th row – not shifted; 1st row – shifted 1 cell to the left; 2nd row – shifted 2 cells to the left; 3rd row – shifted 3 cells to the left

MIXCOLUMNS: Each column is individually transformed linearly (i.e. a known matrix is multiplied to each column).

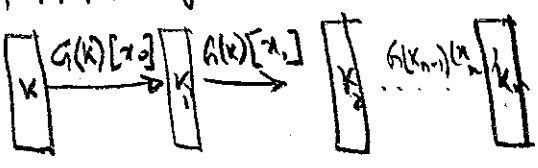
All three operations are easy to implement. The S-block can be computed on the fly or we can store the precomputed values. There's a tradeoff between code size & performance.

AES is supported in processor instructions too. Intel processors have aesinc, aesinclast – one round of aes.

AES does not have any significant attacks. Best key-recovery is 2^{126}. [Both useless attacks from practical POV]
with related keys this possible to recover in 2^{99}.

If we have a PRG we can make a PRF.
Given a secure PRG $G: K \to K^2$ define PRF $F: K \times \{0,1\}^n \to K$

GGM PRF: for $x = x_0, x_1, \ldots x_{n-1} \in \{0,1\}^n$ do → slow (slower than AES)



n invocations of PRG will give us a secure PRF

using Luby-Rackoff we can convert PRF to PRP and get a cipher!

IF I HAVE A PRG I CAN MAKE PRF. PRP.

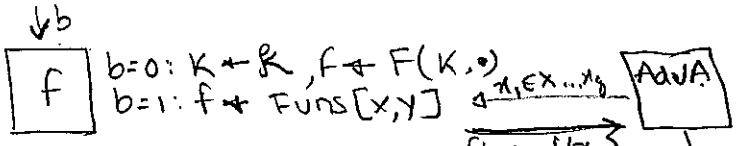
more secure 128 times

MODES OF OPERATION

Week 2 - 4 & 2

PRF - Pseudo random Function, PRP - Pseudo random Permutations

Secure PRF



Adversary cannot distinguish between a random function from X to Y from a function that maps X to Y for a random key

$$Adv_{PRF}[A,F] := \left| Pr[EXP(0)=1] - Pr[EXP(1)=1] \right|$$ is negligible

Secure PRP PRP is an invertible PPF from X to X. Similar definition of advantage.

AES & 3DES are PRPs believed to be secure. Secure PPRs are not necessarily Secure PRFs. This is because PRFs can be non-invertible. However

PRF Switching Lemma: A secure PRP is also a secure PRF is $|X|$ is sufficiently big

More Formally: Let E be a PRP over (K, X), then for any q-query adversary A

$$\left| Adv_{PRF}[A,E] - Adv_{PRP}[A,E] \right| \leq q^2/|X| \times 2$$

1/2 4/4

i.e. for large X if PRP is secure, it can also be used as a PRF.

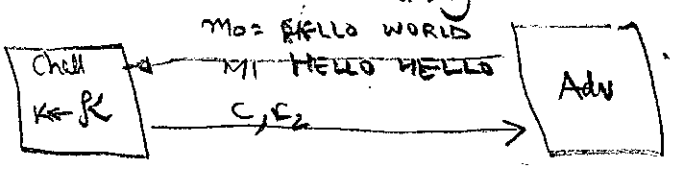
AES is a secure PRP - with a large X size (block) - it can be used as a PRF as well

---

For block ciphers all we have done so far is to define 2 mathematical models PRP/PR and seen 2 practical approximations of these in AES & 3DES. Next we need to see how these primitives are put to use.

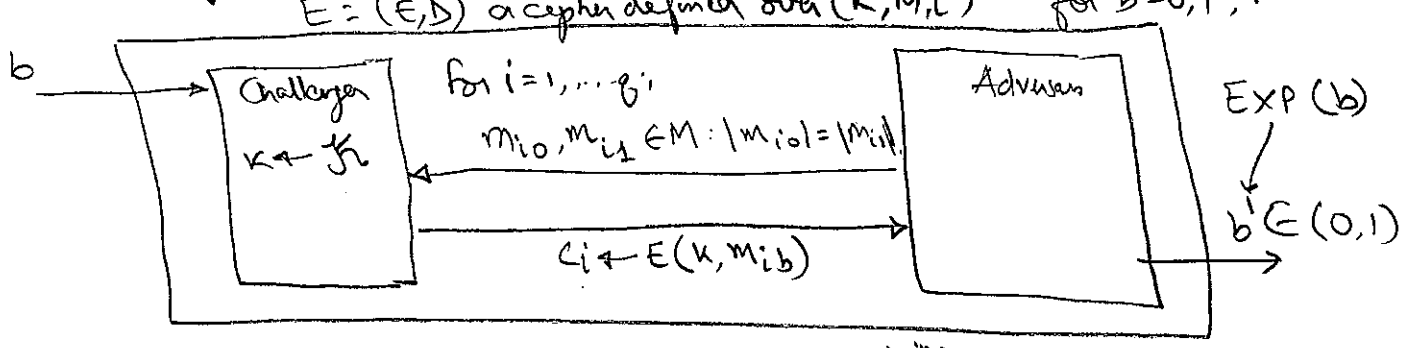
Bad use of PRP (even if it is secure) is ECB (Electronic code book mode)

The key thing here is - messages in real life are longer than the block size of a PRP Given a message that gets broken to more than one 'blocks', what should the key used for encryption be. Can we use the same key to encrypt two blocks of the same message, should we change keys, etc. This questions of this kind that are answered by the "operation" mode. Even if we change the key for every message, just encrypting the blocks in the same message with the same key breaks semantic security.



MODES OF OPERATION FOR MANY TIME KEYS

When we use the same key for multiple messages, we're exposing more information to the adversary. Say an encryption of messages sent over the network., So far our definition of semantic security assumes that the challenger picks a new key for every message pair received from the adversary. If we allow the adversary to run loops of challenges and pick a random key once we're dealing with a "Chosenplaintext adversary". The definition of Semantic Security under CPA is as follows:



The "Game" works as follows. At the beginning the challenger picks 'K'. The challenger can operate in one of the modes, the adversary gives him 2 messages and is consistently returned encryption of m_0 or m_1 based on what mode the challenger is operating in. The objective of the adversary is to output '1' if he believes he's in experiment 1, and 0 if he's in 0.

In CPA the adversary is allowed upto 'q' queries. If he wants to get the cipher equivalent of a chosen plaintext, he can send $m_0 = m_1$ for one of the allowed q queries. With this in mind

E is semantically secure under CPA for all efficient 'A'.

$$Adv_{CPA}[A,E] = \left| P_{r1}[EXP(0) = 1] - P_{r1}[EXP(1) = 1] \right|$$

is negligible

OTP, ECB, DTRCTR mode

Any cipher that encrypts the same message into the same ciphertext is not CPA secure

Attack is Simple: Adv → $m_0, m_0 \in M$, Challenger → $c_0 = E(k, m_0)$
Adv → $m_0, m_1 \in M$, Challenger → c_1, c_0 output 0 if $c = c_0$

Bottomline If the same key is used for more than 1 message
— we must ensure we produce different ciphertext (for the same message)

MANY TIME KEYS

An encryption algorithm is typically deterministic. So given the same input (key, plaintext) it is expected to produce the same output. We've seen that this behavior makes the system insecure semantically under CPA. So the way out is to increase the input parameters and introduce some uniqueness into the input tuple. This is done by the addition of a nonce or a random number that is not repeated.

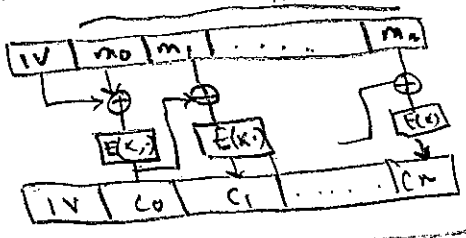
NONCE are only expected to be unique

MTK (Many Time Key) → Randomized Algorithm
→ Nonce Based Encryption (nonces need not be secret or random)

Randomness can be used as a guarantee for uniqueness but for nonce based schemes it is not a requirement. Choosing randomly from a very large space can be used as means of ensuring uniqueness. If uniqueness can be guaranteed by simpler means, like by keeping a counter, the nonce need not be random. If shared state is not possible we use randomized nonces. Next we see two systems that realize these ideas.

CBC : Cipher block chaining: If (E, D) is a PRP $E_{CBC}(K, m)$: choose random IV

CBC with IV — at the beginning we choose a random Initialization Vector, as long as the block size of the cipher.



$$C[0] = E(K, IV \oplus m[0])$$
$$C[i] = E(K, C[i-1] \oplus m[i])$$
$$m[0] = D(K, C[0]) \oplus IV$$
$$m[i] = D(K, C[i]) \oplus m[i-1]$$

q = no. of queries allowed to adversary
practically this is the number of messages we're going to encrypt with K.

ASSUME all is available to adversary. q is the max no of messages before we change key

CBC Theorem: for $L > 0$, if E is a secure PRP over (K, X), then E_{CBC} is semantically secure under CPA over (K, X^L, X^{L+1}).

For a q-query adversary A attacking E_{CBC} there exists a PRP adversary B s.t.:

$$Adv_{CPA}[A, E_{CBC}] \leq 2 \cdot Adv_{PRP}[B, E] + \underbrace{2q^2L^2/|X|}_{\text{error term}}$$

L = max length of message

X^L = input messages of length L

X^{L+1} = output length $L+1$

In words if we start with a secure PRP, the advantage of a CPA adversary is negligible if $\left(\frac{2q^2L^2}{|X|}\right)$ is negligible.

qL = maximum number of blocks encrypted with a single key

for AES $|X| = 128$ bits

assuming $\frac{2q^2L^2}{|X|} \leq \frac{1}{2^{32}}$

$qL \leq 2^{48}$ → if using AES change key after encrypting 2^{48} blocks

Random IV can be replaced by a unique nonce.

However for CBC the nonce must be encrypted with a different key and the encrypted value must be used as the IV.

CBC with nonce

Implementation generally allow non-random nonces to be plugged in.

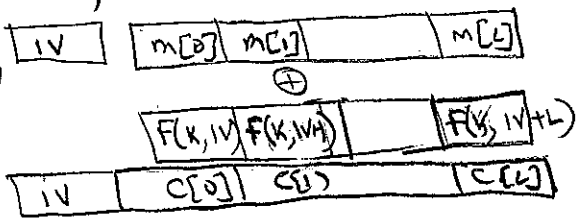
CBC requires padding which can be an overhead

COUNTER MODE

Requires a PRF, in the randomized variant, we choose a random IV.

(non-necessarily a PRP)

$F: K \times \{0,1\}^n \to \{0,1\}^n$



in nonce mode we break the 128 bit IV into 2 parts 64 bit nonce + 64 bit counter.

$$Adv_{CPA}[A, E_{CTR}] \leq 2 \cdot Adv_{PRF}[B, E] + 2q^2L/|X| \leftarrow \text{instead of } L^2, \text{ we have } L$$

INTEGRITY

Integrity without confidentiality. When we care for message content being intact more than them being secret. None of the encryption schemes guarantee that so far. We can always XOR ciphertext with some bits to alter the contents of the message that'll be decrypted and the receiver has no way to validate/detect the changes.

This is achieved by MAC : Message Authentication code.

MAC $I = (S, V)$ defined over (K, M, T) is a pair of algorithms:

(K → key space, M → message space, T → tag space)

S := Signing Algorithm

V := Verification Algorithm.

$tag \leftarrow S(K, m)$, $V(K, m, tag) =$ "yes" or "no"

(sender and receiver share K.)

Consistency requirement → $\boxed{\forall K \in \mathcal{K}, \forall m \in M : V(K, m, S(K, m)) = \text{"yes"}}$, $tag \in T$

(for random environmental errors, not malicious)

Integrity requires a shared key. Using keyless structures like CRC do not guard against adversaries. The attacker can replace the message and CRC simultaneously and fool the receiver.

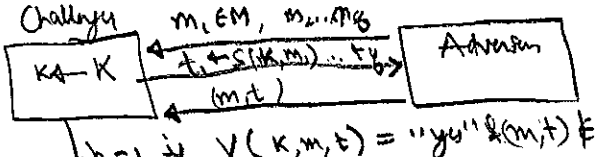
Security: Adversary power : Chosen message attack. The attacker can send arbitrary messages to Alice and ask her to generate tags for them

Adversary's goal : Existential Forgery — Produce a new valid message tag pair.

for (m, t) received generating (m, t') with $t \neq t'$ is bad two

for $m_1, m_2 \ldots m_q$ attacker asks for $t_i \leftarrow S(K, m_i)$

goal to produce $(m, t) \notin \{(m_1, t_1), \ldots (m_q, t_q)\}$



$b = 1$ if $V(K, m, t) =$ "yes" & $(m, t) \notin \{(m_1, t_1) \ldots (m_q, t_q)\}$

$b = 0$ otherwise

$\boxed{|t| \geq \varepsilon}$ (at least ε bits)

$$Adv_{mac}[A, I] = Pr[b = 1]$$

Tags can't be short because existential forgery is possible by simply guessing the tag. For ex. if tags are 5 bits, the adversary can choose any random 5 bit string and emit (m, t). The challenger will output 1 with pr $1/2^5 = 1/32$ (non-negligible)

MACs based on PRF

A secure PRF can be used to make a secure MAC, as long as the output is of sufficient length. For a PRF: $F: K \times X \to Y$

$$S(k, m) = F(k, m) = t$$

$V(k, m, t) :=$ output yes if $F(k, m) = t$.

Formally:

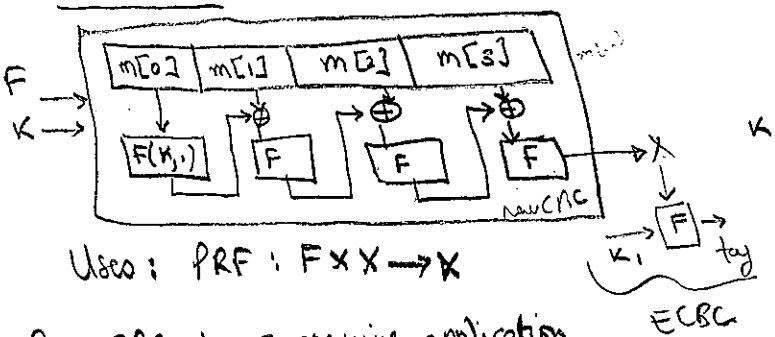
$$\boxed{Adv_{MAC}[A, I_F] \leq Adv_{PRF}(B, F) + 1/|Y|}$$

for ex. for AES, $|Y| = 2^{128}$ & $1/2^{128}$ is negligible, thus if we use AES as a MAC, we're secure, We may not need 128 bits, so we can truncate bits.

The problem however is how do we convert MAC for 16 bytes (AES) to messages longer than 16 bytes.

2 chained constructions

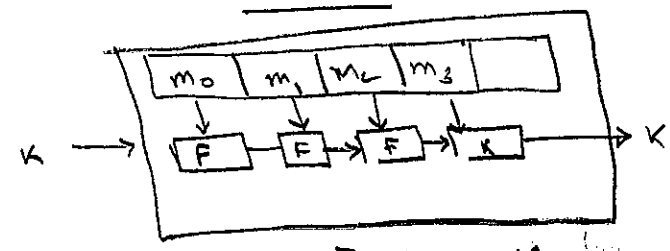
Raw CBC:



Uses: PRF: $F \times X \to X$

Raw CBC is successive application of PRF where input is first xored with the output of last stage.

Cascade:



Uses PRF: $F: K \times X \to K$

Cascade is successive application of PRF where the output of a stage is used as the key for the next stage.

These constructions can be used to make MACs for arbitrary blocks of input.

Encrypted CBC (ECBC-MAC)

Let $F: K \times X \to X$ be a PRP?

define a PRF $F_{ECBC}: K^2 \times X^{\leq L} \to X$

One key and the PRF F is passed to raw CBC and the output is encrypted with the other K to give the tag.

ECBC is insecure without last encryption.

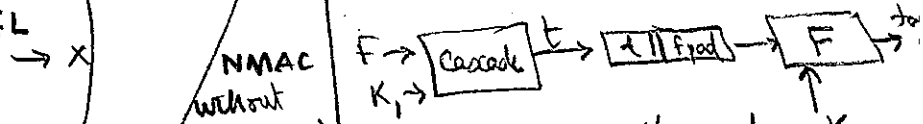
NMAC without last encryption is a bad MAC

output of cascade can be extended.

NMAC

Let $F: K \times X \to K$

$F_{NMAC}: K^2 \times X^{\leq L} \to K$



NMAC is generally used with PRFs that have a block length >> key length. Since the output of cascade is in K we pad it to generate an element in X then apply another F with a different key

ECBC-MAC AND NMAC
Continued

ECBC is insecure if we just use a raw CBC block and don't encrypt its output with an independent key. Here's how the adversary will break the MAC and perform existential forgery.

Adv ⟶ Challenger : [m] (1 block message)
Challenger ⟶ Adversary : m, F(K, m) = t
Adv ⟶ challenger : [m, t⊕m] (2 block message)

rawCBC[K, m]
for 1 block is simply F(K, m)
for 2 block is [m₁, m₂] = F(K, m₁)
F(K, F(K, m₁) ⊕ m₂)

So with 2 messages an adversary can commit existential forgery; given m and t we can emit ([m, t⊕m], t)

NMAC is easier to exploit. Recollect from in NMAC every block is fed into the F with the output of the previous block as the key. So given a tag t for m, it is trivial to construct the tag of m||w (for arbitrary w) as we can generate the tag of the longer message by F(t, w).

In the absence of the extra encryption after the cascade.

Security Theorem: For any L > 0, for every efficient q-query PRF adversary attacking F_{ECBC} & F_{NMAC} there exists an efficient adversary B. such that

$$Adv_{PRF}[A, F_{ECBC}] \leq Adv_{PRP}[B, F] + 2q^2/|X|$$
$$Adv_{PRF}[A, F_{NMAC}] \leq Adv_{PRF}[B, F] + q^2/2|K|$$

in words CBC-MAC is secure as long as $q << |X|^{1/2}$, NMAC as long as $q << |K|^{1/2}$

using AES as our PRF, if we're assuming atleast $1/2^{32}$ as the bounds of neg, for CBC-MAC we have to ensure $\frac{2q^2}{|X|} << \frac{1}{2^{32}} \Rightarrow q << 2^{(128-32)/2} \Rightarrow q < 2^{48}$ / Change key after 2^{48} messages

NMAC we have to ensure $\frac{q^2}{2|K|} < \frac{1}{2^{32}}$ $q \leq 2^{48}$ again.

After signing $|X|^{1/2}$ messages for ECBC & $|K|^{1/2}$ for NMAC, the MACs become insecure.

The PRFs ECBC & NMAC have the following "extension property".

$$\forall x, y, w \quad F_{Big}(k, x) = F_{Big}(K, y) \Rightarrow F_{Big}(K, x||w) = F_{Big}(K, y||w)$$

i.e. in words if I obtain a collision for message x, y I can generate more collisions for longer messages by simply suffixing them with arbitrary messages. So given $|X|^{1/2}$ messages the adversary can attack by generating a forge m||w which will have the same tag as some other.

MAC PADDING

If the message length is not a multiple of the block size the last block needs to be padded. Pads must be invertible, that is given a padded message there should be only one message that it originated from. In other words if 2 messages look the same after padding then we can forge one easily by asking for the tag of 1.

2 Schemes: One simple one is called bit padding, where we keep a 1 at the end of wherever the last block ends and follow it with zeros. While de-padding we ignore every thing upto the last 1. This requires an extra 10^{n-1} block for messages which end at a block boundary.

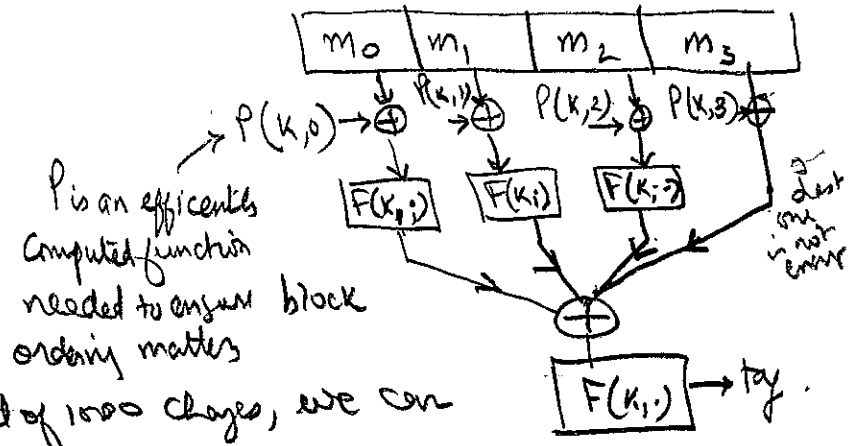
ISO scheme: A variant of the ECBC where the second key is replaced by 2 separate CMAC Keys. The last block is xored with either the first or second "extra" keys if padding is needed for the last block, we zero pad it and xor it with K_1 if not we xor it with K_2. There's no need to do the final encryption on the raw CBC output.

PARALLEL MAC

The MACs we've seen so far are serial, can't be parallelized. PMAC is a construction that allows us to process blocks in parallel.

Let $F: K \times X \to X$ be a PRF

$F_{PMAC}: K^2 \times X^{\leq L} \to X$



P is an efficiently computed function needed to ensure block ordering matters

Each of the blocks computation can be processed in parallel if F is a PRP

F_{PMAC} is incremental. If 1 block out of 1000 changes, we can compute the tag efficiently. Say it m_1 changed to m_1'

$$\text{new tag} = F(K_1, a) \quad \text{where } a = F^{-1}(K_1, \text{tag}) \oplus F(K_1, m_1 \oplus P(K,1)) \oplus F(K_1, m_1' \oplus P(K,1))$$

ONE TIME MAC

Like an OTP, we use a key to do only one MAC. The attacker now only gets 1 (m, t) pair and he needs to produce another to break OTM.

OTMs are secure against <u>all</u> (not just efficient) adversaries and are <u>faster</u>. So if we can realize an OTM, we can use it to MAC long messages much quicker than the PRF-based MACs we've seen so far.

We choose q to be a large prime $(2^{128}+51)$, the key is any 2 numbers (K, a) between 1 & q. The message is a sequence of L blocks, and we imagine these are L coefficient of a L degree polynomial (with the constant term set to 0)

$$P_{msg}(x) = m[L] \cdot x^L + \ldots\ldots m[1] \cdot x$$

now we evaluate $P_{msg}(K) + a \pmod q$ and make this $S(key, msg)$

$$S(key, msg) = (P_{msg}(K) + a) \bmod q$$

V is the same function, followed by check to see if it matches the tag.

As long as we use a key (K, a) for only 1 message this MAC is secure.

Convert a OTM to a many time MAC, by using the <u>Carter-Wegman Construction</u>

Let (S, V) be a secure OTM on $(K_1, M, \{0,1\}^n)$.

Let $F: K_F \times \{0,1\}^n \to \{0,1\}^n$ be a secure PRF

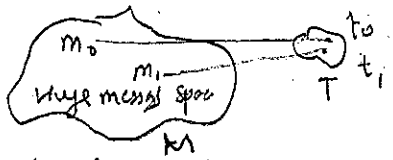
Carter Wegman MAC: $CW((K_1, K_2), m) = r \leftarrow \{0,1\}^n \quad (r, F(K_1, r) \oplus S(K_2, m))$

tag is 2n; slow but short input; OTM; fast long input; m is long

The idea is to apply the OTM on the large input and then XOR it with a random PRF generated value. The verifier is given r, and can efficiently verify the tag.

Every message gets a new r and so we get the best of OTM without its limitations

COLLISION RESISTANCE



Let $H: M \to T$ be a hash function where $|M| >> |T|$.

A collision for H is a pair $m_0, m_1 \in M$ such that $H(m_0) = H(m_1)$ and $m_0 \neq m_1$

H is collision resistant if for all explicit "efficient" algorithms A:

$$Adv_{CR}[A,H] = Pr[\text{A outputs collisions for H}] \text{ is ng.}$$

There are collisions. There is no way to map M to T (with $|M| >> |T|$) without collisions so the requirement of CR is not to show that none exist but to show that there's no efficient & explicit algorithm that can output 2 messages that collide. To break a hash all you have two show is 2 messages that collide.

MACs from CR: Let $I = (S, V)$ be a MAC for short messages of (K, M, T) and $H: M^{big} \to M$. Then we can define $I^{big} = (S^{big}, V^{big})$ over (K, M^{big}, T) as: $S^{big}(k, m) = S(k, H(m))$; $V^{big}(k, m, t) = V(k, H(m), t)$

ex: $S(k,m) = AES_{2\text{-block-cbc}}(K, SHA\text{-}256(m))$ is a secure MAC.

↳ outputs 2x128 blocks. AES is a PRF. So this is a secure MAC.

CR gives us the ability to take constructs that work over small bit sequences and get equivalents for large bit sequences.

BIRTHDAY ATTACK

The Birthday attack is the equivalent of the exhaustive search in block ciphers search. It is a generic attack on all CR functions. in $O(2^{n/2})$ time, for a $H: M \to \{0,1\}^n$ with $|M| >> 2^n$

1) Choose $2^{n/2}$ random messages in M, $m_1, \ldots m_{2^{n/2}}$ (distinct)
2) for $i = 1, \ldots 2^{n/2}$ compute $t_i = H(m_i) \in \{0,1\}^n$
3) Search for collisions $t_i = t_j$. If not found goto 1.

Generally the no. of iterations is very small, so the over all complexity is $O(2^n)$

BIRTHDAY PARADOX

Let $r_1, r_2, \ldots r_n \in \{1, \ldots B\}$ be independent identically distributed integers. What is the # of samples we have to take to have $Pr[r_i = r_j] \geq \frac{1}{2}$

$$Pr[\exists i \neq j \; r_i = r_j] = 1 - Pr[\forall i \neq j \; r_i \neq r_j] = 1 - \underbrace{1}_{r_1 \text{ is unique}} \cdot \underbrace{\frac{B-1}{B}}_{r_2 \neq r_1} \cdot \underbrace{\frac{B-2}{B}}_{r_3 \text{ is distinct}} \cdots \frac{B-(n-1)}{B} = 1 - \prod_{i=1}^{n-1}\left(1 - \frac{i}{B}\right)$$

we know $1 - x \leq e^{-x}$ $\quad \therefore \geq 1 - \prod_{i=1}^{n-1}(e^{-i/B}) = 1 - e^{\frac{-1}{B}\sum_{i=1}^{n-1} i} = 1 - e^{\frac{-(n-1)\cdot n}{2B}} = 1 - e^{-n^2/2B}$

choosing $n = 1.2 \times B^{1/2}$

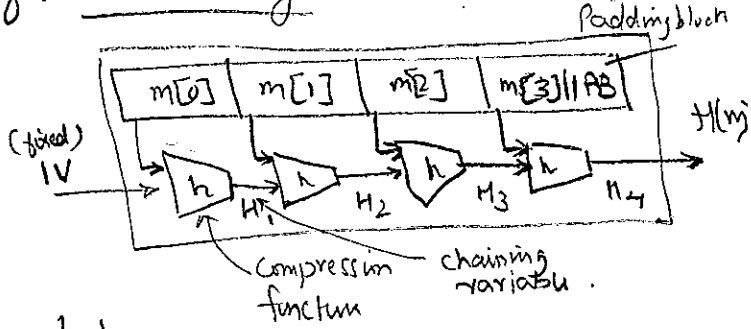
$$Pr[\exists i \neq j \; r_i = r_j] \geq 1 - e^{-0.72} = 1/2.$$

So we only need $\sqrt{B}$ samples before 2 of them collide with $Pr \geq 1/2$.

	key size
SHA1-160	2^{80}
SHA256	2^{128}
SHA512	2^{256}
Whirlpool	2^{256}

MERKEL-DAMGARD PARADIGM

Like the Feistel Network the Merkel-Damgard construction takes a small primitive and makes a more useful primitive. The FN took arbitrary functions and gave us a PRP the MD takes collision resistant functions for small messages and returns a CR function for a large input.



Given $h: T \times X \to T$, DM gives us

$H: X^{\leq L} \to T$

PB: a padding block is essential and has to be present (regardless of whether the input ends on a block boundaries).

$PB := 10000\,[\text{Length}]$ (64 bits)

bit padding

64 bits reserved for message length. We can hash max of $2^{64}-1$ bits in message length

Theorem: If h is collision resistant then H obtained by DM is collision resistant.

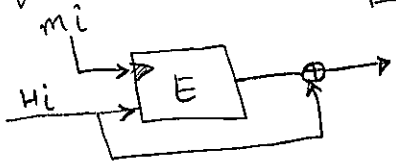
Arguing backwards from the equality of the last 2 outputs, we can prove a collision of H requires either that the two messages are equal or h has collisions. Either of these is a contradiction.

The takeaway is — Given a compression function for small messages we can make one for big messages.

How do we make CF for small inputs.

DAVIES-MEYER CF: CF from a block cipher. $h(H, m) = E(m, H) \oplus H$

Encrypt the chaining variable with message block and XOR with chaining variable



Assuming E is an ideal block ciphers (i.e. $|K|$ random perms from X to X

$E: K \times \{0,1\}^n \to \{0,1\}^n$

Theorem: Finding a collision in DAVIES-MEYER takes $O(2^{n/2})$ time (i.e. as long as it'd take for any CR function using Birthday attack).

⇒ This is as good as it gets (no need for better CR)

MIYAGUCHI-PRENEEL

$h(H, m) = E(m, H) \oplus H \oplus m$ (Whirlpool)

$= E(H \oplus m, m) \oplus m$

and other variants

SHA256 is a MD function using Daives meyer and block ciphers SHACAL2

other obvious variants have problems:

$h(H, m) = E(m, H) \oplus m$ $\left[\begin{array}{l} m' = E(m, H) \\ H' = D(m', m) \end{array}\right.$ given (m, H) collide with

$h(H, m) = E(m, H)$ $\left[H' = D(m', E(m, H))\right.$

HMAC

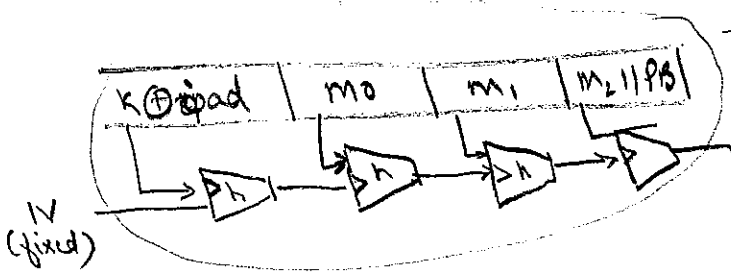
CMAC & ECBC-MAC are MAC constructed using PRFs. Can we make a MAC using just a collision resistant function. Note that a CR function has only one input, i.e. it does not have a key input, we used with PRFs. So we need to combine the key and the message in a way that the output tag is not just CR but also unforgeable.

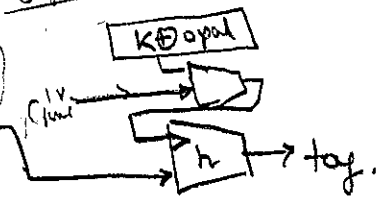
Simply doing $S(K,m) = H(K \| m)$ won't do, if H is a Merkel Damgard hash function, as the attacker can perform existential forgery by computing $H(K \| m \| PB \| w)$.

The standard way to combine the key and message and use them to generate a tag $\longrightarrow$ HMAC (Hash-MAC)

$$\text{HMAC}: S(K,m) = H\left(K \oplus opad \,\|\, H\left(K \oplus ipad \| m\right)\right)$$

output of → $H\left(K \oplus ipad \| m\right)$; fixed 512 bit values specified in HMAC spec.





The construction is same as NMAC with $K_1 = h(K \oplus ipad, IV)$, $K_2 = h(K \oplus opad, IV)$.

Recall that h is a small compression function that is collision resistant

examples: SHA1, SHA256 etc.

AUTHENTICATED ENCRYPTION

Encryption schemes seen so far are not "tamper-resistant". MAC allow us to ensure integrity and encryption allow us to maintain confidentiality. The question next is how do we get both?

How do we combine the two primitives?

If the attacker can modify ciphertext in transit, it is easy to change the meaning of the messages without the receiver knowing.

For CBC with random IV, it is trival to modify the meaning of encrypted messages for 1 block messages by simply XORing the IV (note that we only need to change the IV, not the CT)

Start with your needs

- ONLY INTEGRITY : use a MAC
- ONLY CONFIDENTIALITY: use a block cipher ⇐ AVOID.
- BOTH (which is usually the case): use Authenticated encryption

Authenticated Encryption system ED is a cipher where the attacker can not construct a ciphertext which will be meaningfully encrypted by the receiver. Formally this means:

$$E := K \times M \times \underline{N} \to C \quad \& \quad D := K \times C \times \underline{N} \to M \cup \{\perp\}$$

(N: optional in E, optional in D; $\perp$: rejected cipher.)

The decryption algorithm must be capable of rejecting ciphertext which it believes to not originate from the valid sender. This requirement is called "ciphertext integrity".

∴ An AE system must be semantically secure under CPA & must ensure ciphertext integrity.

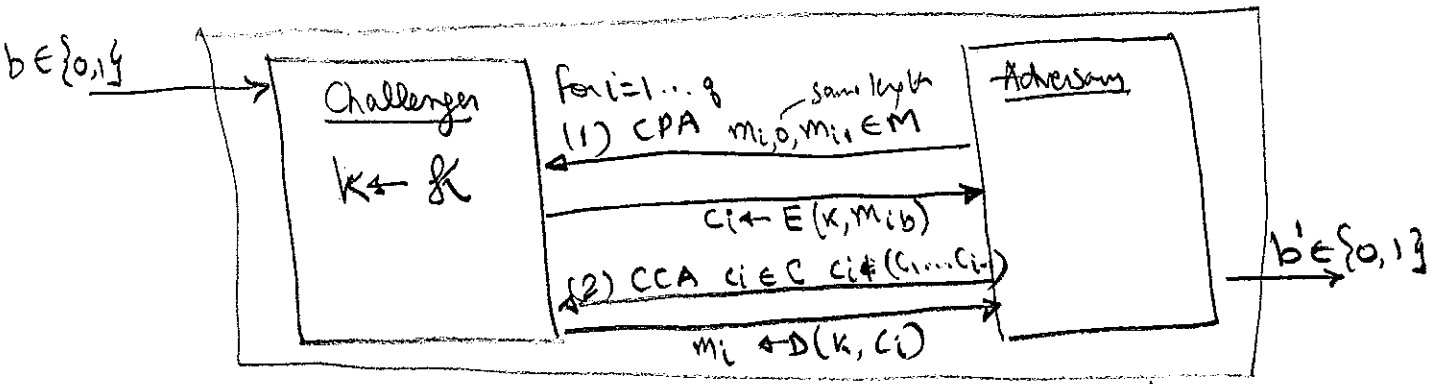
(Everything under CPA ⇐ Any scheme that decrypts everything & does never output $\perp$ is not AE secure.)

CHOSEN CIPHERTEXT ATTACK

Chosen ciphertext security is an extension of the semantic security model that gives more power to the adversary (and is closer to the real world).

Intuitively: in addition to the CPA powers, where the attacker is allowed to submit 'q' arbitrary messages and receive encryptions of those we also allow him to request for decryption of q messages, provided they aren't the ones he issued under CPA. This is CCA (→ chosen ciphertext attack) where we're assuming that the attacker can exploit some weakness in the system to decrypt ciphertexts of his choice.

With this in mind we define the Semantic security game



$$Adv_{CCA}[A,E] = \left| P_r[EXP(0)=1] - P_r[EXP(1)=1] \right|$$

Prove CBC with random IV is not CCA-secure

The goal of all security games is to enable the adversary in guessing the value of b.

$A \rightarrow C$: m_0, m_1, where $|m_0| = |m_1| = 1$ (one block message)

CPA guess [$C \rightarrow A$: (IV, C_0) — C_0 is either $E(m_0)$ or $E(m_1)$ depending on the value of b

CCA guess [$A \rightarrow C$: $(IV \oplus 1, C_0)$ — tweaks the IV (not the same ciphertext)

$C \rightarrow A$: $m_b \oplus 1$ — We know CBC with a random IV produces xor with IV modification

↓

enough to decide if b = 0, or 1.

$Adv_{CCA} = 1$

The point in that CCA gives the attacker the ability to modify known ciphertexts & get them decrypted in ways that can be utilized in attacks.

Authenticated encryption $\xrightarrow[\text{implies}]{}$ CCA: for a q query Adversary A, there exists B_1, B_2 such that

$$Adv_{CCA}[A,E] \leq 2q \cdot Adv_{CI}[B_1,E] + Adv_{CPA}[B_2,E]$$

AE schemes ensure authenticity/authentication of messages because if we assume that the attacker cannot produce valid ciphertexts and if we received a valid ciphertext, we can infer that it must have originated from someone who shares the key. Thus by the assumption of secret shared keys, we can also infer who could have possibly generated the message. However.

AE does not prevent <u>replay attacks</u>, does not account for side channel attacks

---

Combing CPA secure encryption and MAC we can implement AE schemes. Originally there were no "standard" ways to do this. Crypto libraries provide the encryption & MAC primitives and assumed the devs knew how to combine them. That however is non-trivial. Three case-studies:

Assume we have 2 keys K_E (enc key) & K_I → MAC key

3 options: Encrypt <u>and</u> mac (SSH uses this): Given $m \rightarrow E(K_E, m) \,||\, S(K_I, m)$ ← overall ciphertext

this is bad because we're sending tag and tag does not guarantee secrecy. (can leak information about the plaintext.

$E(K_E, m)$ → encryption of message; $S(K_I, m)$ → tag of message

prepared.

Encrypt-then-MAC

IPsec uses this Given $M \rightarrow$ compute $C = E(K_E, m)$ and then tag $S(K_I, C)$ and send both. $C \,||\, S(K_I, C)$

MAC then Encrypt?: Given $m \rightarrow$ Send: $E(K_E, m \,||\, S(K_I, m))$

SSL uses this

OK but use with care, not all ENC + MAC combinations secure

However: with stand-CTR mode or stand-CBC M-then-E is good

Can lead to CCA attacks → Has subtle problems so IPsec method is best.

Standard AE schemes:

* GCM: Uses encrypt then MAC with CTR encryption & CW MAC. Code size ↑
Galois Counter Mode.

CCM: Uses MAC-then-encrypt: CBC mac then CTR mode encryption. code size ↓
AES based (completely implemented using AES) 802.11i

NIST EAX: CTR mode encrypt then CMAC.

All are nonce based. (nonce: non-random but unique)

Standard
All support
AEAD: Authenticated Encryption with Associated data.

Typically comes up in network protocols. We have 2 kinds of data data we want to keep secure and data we want to only ensure integrity for. So we encrypt just sensitive data & MAC every thing. For a network packet we'd encrypt payload and consider headers as only "associated data".

For MAC, we assume that not only can the attacker not produce a forged tag for a message, it can also not produce an 'alternative' tag given a valid one. i.e. 'given' m, adversary shold not produce 't', but even if given (m, t) he should not be able to produce (m, t').
If this happens Encrypt-then-mac will not have ciphertext integrity

OCB: Another AE scheme. Much better than the standard ones. Only evaluates the underlying PRF once per block (Others do it twice, once for MAC once for encryption). Parallelizable. Not used because its patented.

Unless you have a good reason use AES/GCM whenever you need encryption.

Case Study: TLS

TLS: Uses a record protocol for messages exchanged. Records are at-most 16 KB. Uses Unidirectional keys for encrypted records, and uses stateful encryption: This means:

established using a key exchange { – both browser and server share a pair of keys ($K_{s \to b}$ (server to browser) & $K_{b \to s}$ (browser to server))

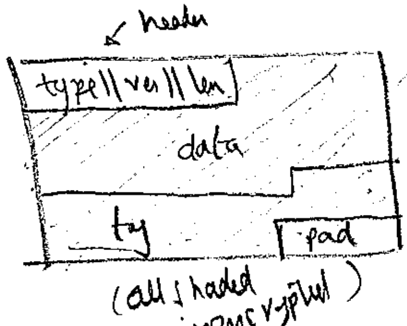
– both browser & server maintain 2 64bit counters $Ctr_{b \to s}$ & $Ctr_{s \to b}$ — prevent replays.

Counters are initialized to 0 at the start of the session and incremented at every packet send.

TLS supports a range of algorithms. Mandatory support for AES-128 CBC for enc & HMAC-SHA1 for hashing.

Each of the uni-directional keys is actually a pair of keys; one for encryption & the other for MACing. TLS uses MAC then encrypt.

Encrypting side: def enc($K_{b \to s}$, data, $Ctr_{b \to s}$):



1) Compute tag(t): $S(K_{mac}, [++Ctr_{b \to s} \,||\, header \,||\, data])$ — counter incremented to ensure every record gets a unique #
2) Pad [header || data || tag] to AES block size. — Using PKCS padding
3) CBC Encrypt with K_{enc} with new random IV — TLS 1.1 fixed (BEAST Attack); older versions used chained IVs
4) Prepend header

Decryption: dec($K_{b \to s}$, record, $Ctr_{b \to s}$):

1) CBC decrypt using K_{enc}
2) Check pad format. See if it is well formed. If not return bad_record_mac
3) Check tag on [$++Ctr_{b \to s}$ || header || data] (increment counter). If no match send bad-record-mac (same)

If you don't do that — all errors are responded with the same response.

TLS 1.1 is good!

older version
- used chained IV — beast attack
- Padding oracle attacks by distinguishing between padding errors & decryption errors. they leaked information.

802.11b WEP bad example of AE system.

uses CRC instead of a proper MAC.

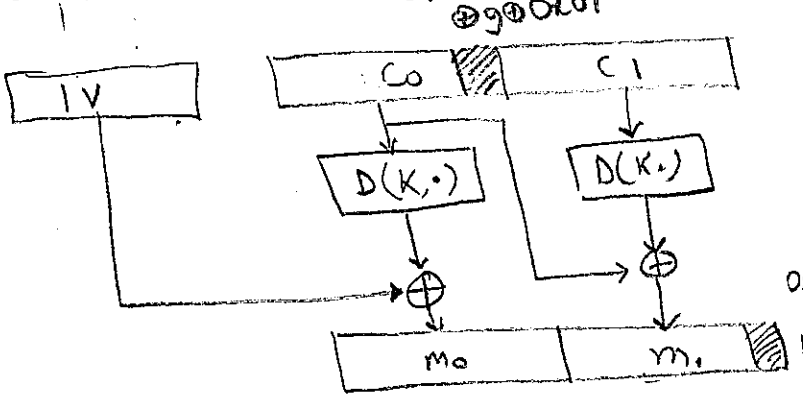
CRC is linear $\forall m, p \quad CRC(m \oplus p) = CRC(m) \oplus F(p)$

so attacker can XOR both CT & CRC so that the message is changed

CBC PADDING ATTACKS

AE provides CPA security + Ciphertext integrity. But it needs to be implemented right. We should use GCM, (or other standard ways). Non standard implementations can have issues. Even TLS had them. TLS before 1.1 could be attacked using the "padding oracle" attacks. (ONLY IF CBC mode is used, not with CTR modes because CTR do not use padding). The problem was that the protocol required different response for padding format errors & encryption error. These can be used to expose the encrypted plaintext. How?

Consider the CBC decryption circuit: Imagine we have a server that responds differently to decryption failure & padding format error. And that we want to uncover m_1.



We do that by forcing the decryption of m_1 with bad padding.

Consider how the last byte of m_1 is obtained.

$$m_1 = D(c_1) \oplus c_0 \quad \text{so if we tweak } c_0 \text{ last byte to be}$$

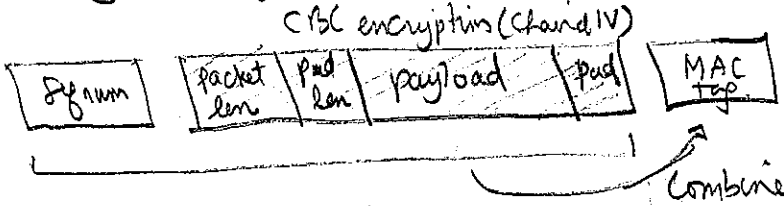
$$c_0[\text{last byte}] = c_0[\text{last byte}] \oplus 0x01 \oplus g \quad \text{where } g \text{ is our guess of what the last byte of } m_1 \text{ is.}$$

if our guess is right: upon decryption the last byte of m_1 will be received as 0x01. This is a valid pad, so we don't expect to see a padding-error response. If not the last byte will get decrypted to something else which is most likely a bad pad and will raise a padding error response. We can use this feedback to try various possibilities of g. In 256 attempts we'll discover 1 byte of the plaintext.

Next we take 2 bytes of c_0 and xor them to ensure the produce 02 at the end. For the last byte this is easy by XOR with $m[\text{last}] \oplus 2$ for second last we try again 256 values. Thus in 16×256 attempts we can complete discover the last block

CASE STUDY - SSH

SSH uses MAC and Encrypt. There's a key exchange and after that there's a "record" protocol that is used.



Decryption Logic:

i) Decrypt just packet length
ii) Read as many packets off the network as specified by length
iii) Decrypt remaining ciphertext blocks
iv) Compute MAC, and respond with error if tag fails.

There are multiple problems with the scheme: CBC with chained IV — MAC sent in clear, can expose information about PT.

However here, we're interested in an attack which is made possible by the use of "non-atomic decryption" - The length field is decrypted first, so if the attacker captures a packet and resends the same with a new seq-no. to the server. The server will attempt to read the decrypted # of packets off the network. Effectively, the server will leak the attacker the decryption of the first 32 bits of the CT..

Using non-atomic decryption is bad! It'd be better if length was sent in plaintext

KEY DERIVATION

How to derive many keys from one key? Typically random generation or Key exchange gives us one good key, but we need multiple keys. This is what Key Derivation functions do

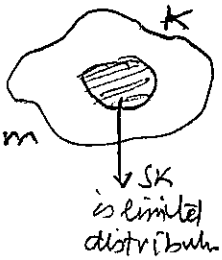
If the source key is uniform, we can use a PRF in counter mode, using a PRF as a PRG.

source key ↑ SK, application identifier (context) ↗ CTX, L

EXPAND ⇒ KDF(SK, CTX, L) := F(SK, (CTX||0)) || F(SK, (CTX||1)) || ..... ||F(SK,(CTX||

↓ length.

The PRG made from PRF is only good if SK is uniform. Generally this is not the case. In such cases we need to **Extract-then-expand**, and that requires a salt. A salt is used to make a non-uniform key, uniform



Salt = publicly known, fixed string chosen at random

Step1: Extract a pseudo random key from the source key. Step2: Expand using a PRF

HKDF = KDF made from HMAC

- extract use K ← HMAC(salt, SK)
  - salt → HMAC Key, SK → HMAC data
- expand use HMAC as PRF with Key as K

Generally after handshake the key is run through HKDF to generate real session keys.

ONLY USED WHEN SK in non-uniform but high entropy.

PBKDF

Password have insufficient entropy.

PKCS#5 = $H^{(c)}$(pwd||salt)

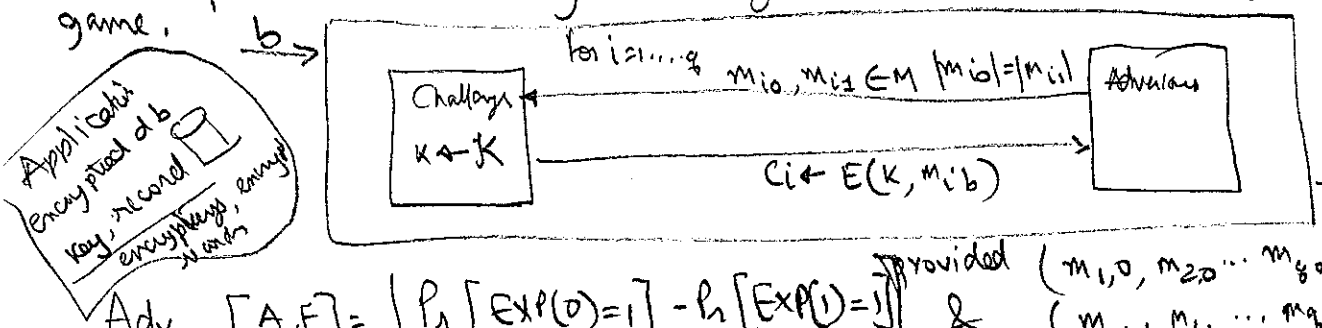
↓ Hash function / made slow by iterations

Basically PBKDF is a "defense" to make it harder to break the lack of entropy in the key. By repeating Hash function, we make it difficult for the attacker to try out all possible passwords (dictionary attack)

DETERMINISTIC ENCRYPTION

Encryption without nonces. The same value gets encrypted to the same ciphertext. This is the very same thing we wanted to avoid using chaining. (ECB) Actually such deterministic encryption is not CPA secure. However it is still useful if we have additional guarantees about the message space.

If we know that the encryptor guarantees never to repeat the same message then we're good. Assume we're encrypting the unique "user-id" which are unique across all encrypted messages, we're in turn tightening our usual CPA game.



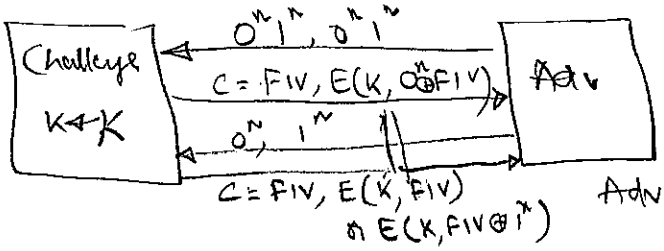
$$Adv_{dCPA}[A,E] = \left| Pr[EXP(0)=1] - Pr[EXP(1)=1] \right|$$ provided $(m_{1,0}, m_{2,0} \cdots m_{q,0})$ are distinct & $(m_{1,1}, m_{2,1} \cdots m_{q,1})$ are distinct.

is negligible.

Intuitively if no two messages issued by the Adv are same and we are semantically secure. Then we're deterministically CPA secure.

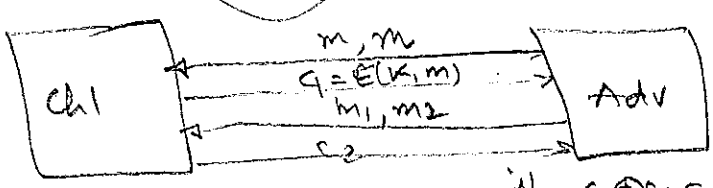
Candidate 1: CBC with fixed IV is not determ. CPA secure

Common mistake: Assuming fixing IV makes CBC deterministic



Note that the requirement is that all 1st messages are unique.

CTR with fixed IV is not det CPA secure too. Because it reduces to 2 time pad. CTR is actually a bad choice for det. encryption



We need special "modes" for det. Encryption.

if $c_1 \oplus c_2 = m \oplus m_2$ output 1
else output 0.

Modes + Constructions for Deterministic Encryption

Needed for maintaining encrypted databases where we want to query records over an encrypted index.

Synthetic IV (SIV): Specially useful if messages are > cipher block size

Let (E, D) be a CPA secure encryption $E(K, m; r) \to C$ ex: AES CTR/CBC mode when $r = IV$

& let $F: K \times M \to R$ be a secure PRF

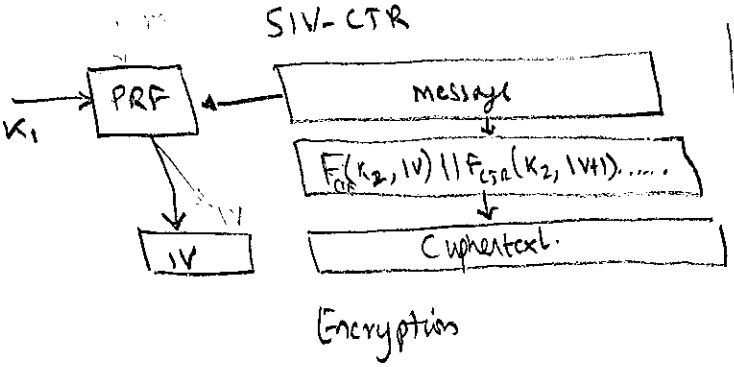
$$E_{det}((K_1, K_2), m) = \begin{array}{l} r \leftarrow F(K_1, m) \\ C \leftarrow E(K_2, m, r) \end{array} \quad \text{output } C$$

Intuitively → we use the PRF to generate a unique random message from the message m, and then use it with E to generate the CT. Since the PRF is secure 'r' is unique for m and we end up with the requirement of det encryption

Note that the complete ciphertext = $r \| C$ (i.e. we need to output r along) this is why its not suitable for short messages (because it increases CT size)

An interesting by product of SIV is that the extra "r" can serve as a message tag. We get ciphertext integrity for free.

SIV is DAE (Deterministic authenticated encryption). Consider SIV-CTR



We decrypt and then run the plaintext through the PRF to get the tag. If this is the same as the IV we used in the CTR mode the message is authentic.

if F is a secure PRF and CTR derived from F_{CTR} is CPA-secure then SIV-CTR from F is DAE. — It is not possible for the adversary to produce a CT that decrypts to something that gives the same value that was used as the IV in CTR mode. This is what Ciphertext integrity requires.

IF the records you're encrypting are long and you don't mind the extra IV → SIV is good. if not we need a different scheme.

[DETERMINISTIC ENCRYPTION FOR SHORT MESSAGES]

If (E, D) is a secure PRP, simply encrypting them using PRP is sufficient for deterministic CPA. The trouble is what if the message is longer than block size. One option is to use SIV, the other is to generate a N bit prp from a n bit PRP. The construction is called EME. pretty complicated.

The other trouble with just using a PRP is that we do not get DAE there's no integrity. Integrity can be added by simply appending 0s at the end. Just ensure that the no. of 0s is enough to have $1/2^n$ as negligible.

≤ 128 bits or AES block size.

Takeaway: If you have very short messages (say record-ids) that are unique, just suffix 80 or more 0s at the end. and use AES to encrypt it.

If you can't fit your data into an AES block use SIV.

[TWEAKABLE ENCRYPTION]

Consider an application for encrypting disk sectors. If a sector is 4KB, and there's only 4KB of sector space to store it, we cannot afford randomized encryption schemes (Like CBC/CTR) because there's nowhere to store the random input (IV, etc). We need to use a scheme where encryption is not 'expanding'. Note that we don't have room for integrity tags too. We need a PRP

Just a PRP is not going to work, because we're not dealing with det. encryption where we're guaranteed that messages are not repeated. Here blocks on the disk and sectors can be repeated. So we need to change keys for every sector. One scheme is to use a PRF and generate keys using the sector number and a master key. $K_t = PRF(K, t)$ ← key for sector t

TWEAKABLE BLOCK CIPHER

Continuing with the example of disk encryption: The problem is to generate multiple keys (so that they're different) for each sector/block. One construction is to use the sector number as the "tweak" that is used to generate the key. Theoretically, this can be modelled by a TWEAKBLE BLOCK CIPHER. The goal is to take one PRP and construct many out of it.

$E, D : \mathcal{K} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X}$, such that for every $t \in \mathcal{K}$ & $k \in \mathcal{K}$ $E(k, t, \cdot)$ is an invertible funct. indistinguishable from random!

($\mathcal{K}$: key space; $\mathcal{T}$: tweak space; $\mathcal{X}$: message space)

A simple construction for this would be, assuming $K = X$ (like AES)

$$E_{tweak}(K, t, x) = E\left(E(k, t), x\right)$$

(encrypt tweak and use the encryption as the key to encrypt x.)

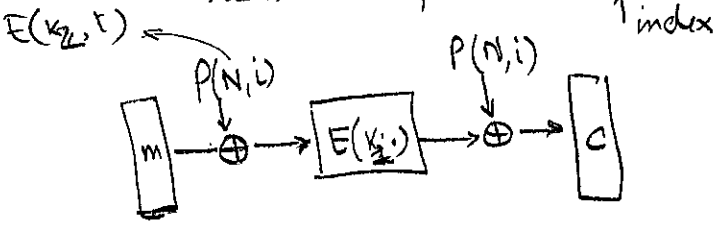
Assuming we have n blocks, so n tweaks this simple tweakable cipher will involve $2 \times n$ invocations of E. Can we do better?

YES: XTS:

Let E, D be a secure PRP, $E : K \times \{0,1\}^n \to \{0,1\}^n$

XTS: $E_{tweak}\left((K_1, K_2), (t, i), x\right)$

(takes 2 keys and splits the tweak into 2 parts. t & i ↑ index)



Step 1: compute $N := E(k_2, t)$

Step 2: (repeat for each block)

for block i: Compute $m_i \oplus P(N, i)$, say P_i

Compute $C = E(K_1, P_i) \oplus P(N, i)$

So to encrypt n blocks, we need $n+1$ encryptions. P is a efficiently computable function that is negligible w.r.t. E.

So – in general the idea is to prefix & suffix the encryption with a tweak that can ensure different blocks. Need to ensure that the tweak is used properly. Suppose we use t instead of $N \Rightarrow E\left(K_1, (m \oplus P(t,i))\right) \oplus P(t,i)$. This is bad because the tweak is not "randomized by the key" and can be used to break the scheme.

$$E(K, (t,1), P(t,1)) \oplus E\left(K, ((t,1), P(t,2))\right) = P(t,1) \oplus P(t,2).$$

FORMAT PRESERVING ENCRYPTION

Given a PRF like AES $F: K \times \{0,1\}^n \to \{0,1\}^n$ and given $s: 0 < s \leq 2^n$
Construct a PRP on $\{0,1,\ldots,s-1\}$.

Typical application: encrypting a credit card number. A credit card number is effectively a 2^{42} bit number. If we encrypt it using AES, we'd effectively convert a 42 bit number to a 128 bit ciphertext. This may be a problem, if the overall application requires the encrypted data to be transmitted across components that expect to see a credit card number (and would break if we pass a 128 bit number). So we need a scheme that encrypts a 42 bit number to another 42 bit number. Note that the output size does not effect the security of the scheme, we can still use a 128 bit key if we like. Such encryption is called "format preserving encryption".

FPE Schemes: Step 1: find the smallest power of 2 that is capable of representing all possible values. say S.

: Find a PRF capable of outputting $\leq b$ bits. Use AES and truncate output to $S/2$ bits

: Use a 7 stage Feistel Network (3 required by Luby Rack to convert the PRF into a PRP.

Step 2: Repeatedly apply the PRP to obtain a s bit value that fits inside your acceptable range.

BASIC KEY EXCHANGE

We know how to implement security primitives as long as we have shared secret key. The next question is how are these shared secrets established. One trivial / impractical solution is to require every pair of users to share a secret key. This requires $O(n^2)$ keys, and is hard to manage.

The other alternative is to have an online trusted third party. Kerberos is an example. Everyone shares a key with the TTP and TTP generates and issues session keys for interactions between users (as tickets). While this works well (specially in an enterprise setup, where we can have a trusted server) there are issues with using this when TTP could be breached or not trusted.

"Can we generate shared secrets without a TTP?" → Fundamental question in Public Key cryptography.

MERKEL PUZZLES: First attempt at P-K crypto. Ralph Merkel in 1974 presented the idea as a part of his undergrad seminar. The prof. did not grasp the significance of the finding and it was only 5 years later that it was published.

The idea is the basic idea of PKC: Make the attacker do a lot more work than the valid parties.

Merkels puzzles use only shared key primitives. They make a very inefficient protocol and is practically useless. "Puzzle" is a solvable problem which requires work.

Example: Breaking a 32 bit cipher. We can brute force a 32 bit key, but it'll take some time, so this makes it a puzzle.

Attacker will have to break all 2^{32} puzzles to get the key that corresponds to x_i ⇒ $O(2^{64})$

Merkles protocol uses this same puzzle.

Alice:
- Generate 2^{32} encryption puzzles:

for $i = 1 \ldots 2^{32}$ chose random $P_i \in \{0,1\}^{32}$ and $x_i, K_i \in \{0,1\}^{128}$

set puzzle $i = E_{AES}(0^{96} \| P_i, \text{"PUZZLE\#}x_i\text{"} \| K_i)$

Work: Assuming constant time for generation. Alice does $\Theta(n) = O(2^{32})$ work

- Send 2^{32} puzzles to Bob.

Bob: Pick a random puzzle from the 2^{32} received and break it (by trying all 2^{32} possible keys"). Bob does $O(2^{32})$ work. Send x_i of the broken puzzle to Alice, and use K_i as the key

Attacker does double the work

DIFFIE-HELLMAN PROTOCOL

Merkel puzzles are said to have a "quadratic gap". This means that the participants do $O(n)$ work and the attacker does $O(n^2)$ work. So the gap is quadratic. Can we increase this gap. No → if we use only shared key primitives. We need to use more algebraic primitives. Public key systems achieve an "exponential gap".

Diffie-Hellman:

Fix a large prime p (600 digits prime) ≈ 2000 bits, choose a g in $\{1, \dots p\}$ ← generator

p & g are parameters of DH exchange.

Alice: choose random a in $\{1, \dots p-1\}$

Bob: choose random b in $\{1, \dots p-1\}$

$A \longrightarrow B: g^a \pmod p$

$B \longrightarrow A: g^b \pmod p$

Alice computes $(g^b)^a$ ← what Alice chose; g^b ← what B sent

Bob computes $(g^a)^b$

then $K_{AB} = g^{ab} \pmod p$

DH is insecure against MITM.

DH can be used non-interactively by if everyone publishes g^a, any one can send a secure message by computing g^{ab}

Why can't the attacker compute $g^{ab} \pmod p$.?

What does he see: $p, g, g^a \pmod p$ & $g^b \pmod p$ where p is a large prime

Can he compute a function → $DH_g(g^a, g^b) = g^{ab} \pmod p$.. General Number Field Sieves

Yes but it takes terribly long. The fastest known algorithm for this is GNFS

if p is n bits long GNFS = $e^{\tilde{O}(\sqrt[3]{n})}$ ← sub exponential. roughly.

Comparisons: p = 1024 bit GNFS takes as much time as brute forcing 80 bit keys

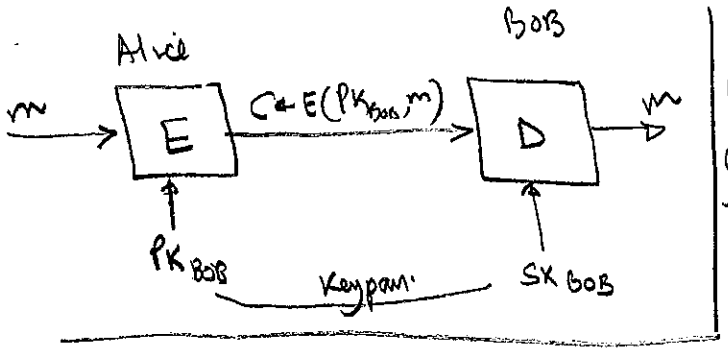
160 bits	p=1024 ⇔	80 bit cipher.
256 bits	p=3072 ⇔	128 bit cipher
512 bits	p=15360 ⇔	256 bit cipher.

generally p = 2048 bits.

Security of the key exchange protocol must be comparable to the security of the block cipher.

DH is more a template and we can replace the algebraic modulo arithmetic with the Elliptic curve computations and those require much better parameters.

PUBLIC - KEY ENCRYPTION



A PK system is a triple of algorithms (G, E, D)

where G: randomized algo outputs a keypair (PK, SK)

Generation

$E(PK, m)$: randomized algo that takes $m \in M$ and outputs $C \in C$.

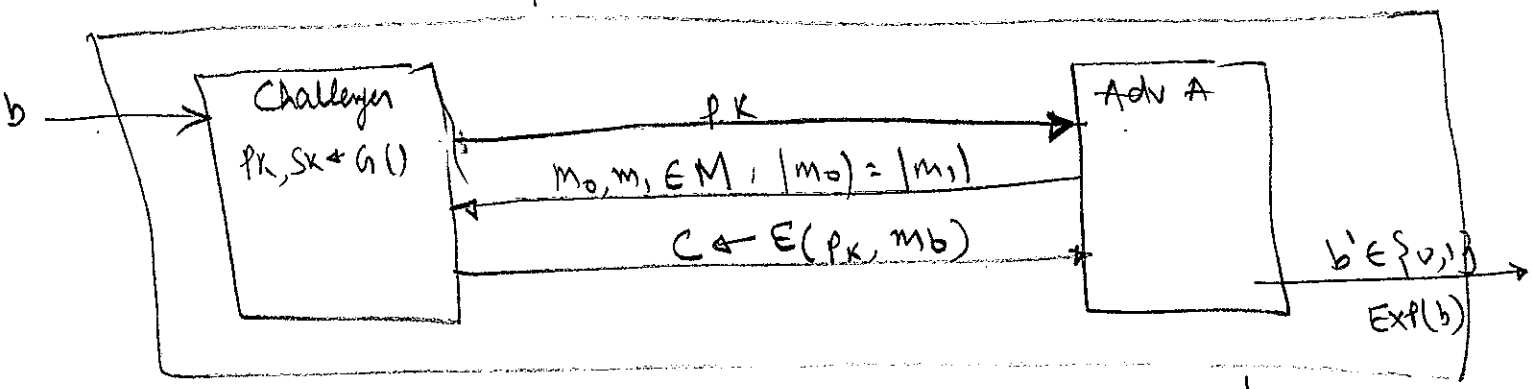
$D(SK, C)$: det algorithm that takes $C \in C$ and outputs $m \in M$ or $\perp$.

$\forall (PK, SK)$ output by G

$\forall m \in M: D(SK, E(PK, m)) = m$

SEMANTIC SECURITY

$E = (G, D, E)$ is semantic secure (IND-CPA) if for all efficient A:



$$Adv_{SS}[A, E] = \left| Pr[EXP(0) = 1] - Pr[EXP(1) = 1] \right| < \text{negl}$$

CPA is always given in PK systems, so ideally the game doesn't require a message to be sent to the challenger for encryption (generation of ciphertext)

Number Theory

Notation: N is a positive integer. 'p' is a prime

ring $\rightarrow \mathbb{Z}_N = \{0, 1, 2, \ldots, N-1\}$ ex. $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$

We can define arithmetic operations over $\mathbb{Z}_N$ and this would be called arithmetic modulo N. Arithmetic in $\mathbb{Z}_N$ follows the same laws of arithmetic.

ex: $9+8=5$ in $\mathbb{Z}_{12}$, $5\times 7 = 11$ in $\mathbb{Z}_{12}$, $5-7=10$ in $\mathbb{Z}_{12}$.

GCD: Greatest common divisor. for all / integers $x, y \rightarrow$ there exist integers a, b such that $ax+by = \gcd(x,y)$ (extended euclid's algorithm) $\rightarrow$ GCD of 2 numbers can be expressed as a linear combination using a, b.

if $\gcd(x,y)=1 \rightarrow x$ and y are relatively prime.

Modular Inversion: Element of x in $\mathbb{Z}_N$ is an element y in $\mathbb{Z}_N$ such that $x \cdot y = 1$ in $\mathbb{Z}$

ex: Consider $\mathbb{Z}_5$, inverse of 2 in $\mathbb{Z}_5$ is 3, because $2\times 3 = 1$ in $\mathbb{Z}_5$.

More generally inverse of 2 in $\mathbb{Z}_N$ with N odd is $\Rightarrow \frac{N+1}{2}$

Not all elements have inverses in $\mathbb{Z}_N$. x in $\mathbb{Z}_N$ has an inverse iff $\gcd(x,N)=1$

ex: 2 in $\mathbb{Z}_5$ has an inverse 3, because 2, 5 are relatively prime.

3 in $\mathbb{Z}_6$ has no inverse $\gcd(3,6)=3$. $\{0,1,2,3,4,5\}$

$3\times 0=0$, $3\times 1=3$, $3\times 2=0$, $3\times 3=3$, $3\times 4=0$, $3\times 5=3$

Proof:

if $\gcd(x,N)=1$, inverse exists

$\gcd(x,N)=1 \Rightarrow a\cdot x + b\cdot N = 1 \Rightarrow a\cdot x = 1 \mod(N)$

$\boxed{x^{-1} = a}$ ← mod N ($b \cdot N$ can be dropped $=0$)

using euclidean extended algo.

only if say $\gcd(x,n) > 1$, inverse cannot exist.

Solve $ax+b=0$ in $\mathbb{Z}_n$

$x = -b\cdot a^{-1}$

Ext euclidean algo $O(n^2)$ to n bit N

$\mathbb{Z}_N^*$ ⇒ Set of elements in $\mathbb{Z}_N$ that have an inverse.

$\mathbb{Z}_N^* = \{x \in \mathbb{Z}_N : \gcd(x,N)=1\}$ ex. $\mathbb{Z}_{12}^* = \{1,5,7,11\}$

for Prime p $\mathbb{Z}_p^* = \mathbb{Z}_p / \{0\} = \{1,2,\ldots,p-1\} \Rightarrow |\mathbb{Z}_p^*| = p-1$

Modular Arithmetic Cont
Fermat & Euler

Fermat: Let p be a prime $\quad \forall x \in \mathbb{Z}_p^* \quad x^{p-1} = 1$ in $\mathbb{Z}_p$

This can be used to compute inverses in $\mathbb{Z}_p$ (note only mod primes, unlike Euclids which works for composites)

we know $x^{p-1} = 1 \Rightarrow x \cdot x^{p-2} = 1$ in $\mathbb{Z}_p \Rightarrow x^{-1} = x^{p-2}$

So by fermat's theorem, we can compute the inverse by simply raising x to $p-2$

However, Euclids extended algo is more efficient. This takes $O(\log^3 p)$ euclid $O(\log^2 p)$

Another application is prime number generation.

Say we want to generate a prime p of length 1024 bits, we

i) choose a random integer $p \in [2^{1024}, 2^{1025}-1]$

ii) See if $2^{p-1} = 1$ in $\mathbb{Z}_p$

if yes output p; else repeat

p outputted is not guaranteed to be prime by $Pr[\text{Prime}] < 2^{-60}$

Euler $\mathbb{Z}_p^*$ is a cyclic group, that is $\exists g \in \mathbb{Z}_p^*$ such that

$$\{1, g, g^2, \ldots g^{p-2}\} = \mathbb{Z}_p^*$$

→ note that g^{p-1} is not needed, it is 1 by fermats theorem

i.e. there exists a generator g such that all elements of $\mathbb{Z}_p^*$ can be obtained by multiplying g with itself.

Consider $p=7$, we can see that all elements in $\mathbb{Z}_7^*$ can be obtained by raising 3 to powers from 0 to 5. $\mathbb{Z}_7^* = \{1,2,3,4,5,6\} = \{3^0, 3^2, 3^1, 3^4, 3^5, 3^3\}$

Not every element is a generator, but if p is prime Euler says there is at least one generator

for ex. 2 in $\mathbb{Z}_7^*$ will only generate $\{1,2,4\}$

for $g \in \mathbb{Z}_p^*$ the set $\{g^0, g^1, g^2 \ldots\}$ is called the group generated by g $\quad \langle g \rangle$

the order of g in $\mathbb{Z}_p^*$ is the size of $|\langle g \rangle|$. if $|\langle g \rangle| = |\mathbb{Z}_p^*|$ then g is a generator.

$= (\text{smallest } a > 0 \text{ s.t. } g^a = 1 \text{ in } \mathbb{Z}_p)$

$\text{ord}_7(3) = 6$ ← $7-1 = (p-1)$ ← 3 is a generator

$\text{ord}_7(2) = 3$

$\text{ord}_7(1) = 1$

$\text{ord}_{35}(2) =$

$\forall g \in \mathbb{Z}_p^* \quad \text{ord}_p(g)$ divides $p-1$ → Lagrange

Euler generalized fermats result ($x^{p-1}=1$ for $x \in \mathbb{Z}_p^*$), by using a φ function

$\varphi(N) = |\mathbb{Z}_N^*| \Rightarrow \varphi(N)$ = no. of elements in $\mathbb{Z}_N^*$ (note that N is not necessarily prime)

ex: $\varphi(12) = |\{1,5,7,11\}| = 4$, $\varphi(p) = p-1$ ← for primes.

if $N = p \cdot q$ (product of 2 primes) $\quad \varphi(N) = N - p - q + 1 = (p-1)(q-1)$

p: # of multiples of q; q: # of multiples of p

$$\forall x \in \mathbb{Z}_N^* : x^{\varphi(N)} = 1 \text{ in } \mathbb{Z}_N$$

for primes $\varphi(p) = p-1$ and this theorem reduces to Fermats theorem.

Applications: what is $2^{10001} \bmod 11 \Rightarrow (((2^{10})^{10})^{10}) \cdot 2 = 2$.

↑ 1 because 2^{11-1} in $\mathbb{Z}_{11}$.

$2^{245} \bmod 35 \Rightarrow 2^{240} \cdot 2^5 = (2^{24})^{10} \cdot 2^5 = 32$.

↑ 1 because $2^{\varphi(35)} = 1$ in $\mathbb{Z}_{35}$

Using Euclids algorithm we can solve linear equations in $\mathbb{Z}_N$

ex. $ax+b=0$ in $\mathbb{Z}_N \rightarrow x = -b \cdot a^{-1}$ in $\mathbb{Z}_N$

What about higher degree polynomials. Specifically polynomials in $\mathbb{Z}_p$ of a simpler form $x^n - c = 0$ in $\mathbb{Z}_p$ generally the solution would involve finding the n^{th} root of c in $\mathbb{Z}_p$.

Let p be a prime an $c \in \mathbb{Z}_p$, then $x \in \mathbb{Z}_p$ such that $x^e = c$ in $\mathbb{Z}_p$ is called the e^{th} root of c.

ex. $7^{1/3} = 6$ in $\mathbb{Z}_{11}$ because $6^3 = 216 = 7$ in $\mathbb{Z}_{11}$

$3^{1/2} = 5$ because $5^3 = 25 = 3 \bmod \mathbb{Z}_{11}$

Not all e^{th} root exist.

Solving Modular Quadratic Equations

Let p be a prime and $c \in \mathbb{Z}_p$, we say x is the e^{th} root of c if

$$x \in \mathbb{Z}_p \quad \text{and} \quad x^e = c \text{ in } \mathbb{Z}_p$$

Not all e^{th} roots exist, for example $2^{1/2}$ does not exist in $\mathbb{Z}_{11}$. The question then is → When do e^{th} roots exist? How do we find them?
There is no general algorithm to find modular e^{th} root in $\mathbb{Z}_N$, we walk over special cases. (Firstly, we're only dealing with $\mathbb{Z}_p$ ← primes)

Case 1: e is relatively prime to $p-1$, ex. cube root in $\mathbb{Z}_{11}$ (3, 1 are relatively prime)
if $\gcd(e, p-1) = 1$. then every element $c \in \mathbb{Z}_p^*$ has an e^{th} root that is easy to compute.

(11th root of 2 in $\mathbb{Z}_{19}$ = 13)

Since $\gcd(e, p-1)$ e has an inverse in $\mathbb{Z}_{p-1}$ lets call it d.
then $\boxed{c^{1/e} = c^d}$ that is the e^{th} root is $c^{e^{-1}}$

since $e \cdot d = 1$ in $\mathbb{Z}_{p-1}$ if c^d is e^{th} root $(c^d)^e = c^{de} = c$

If you want e^{th} root and e is relatively prime to $p-1$, then c^d is the e^{th} root, where d is the inverse of e in $\mathbb{Z}_{p-1}$ ← (not in $\mathbb{Z}_p$)

Case 2: Consider square root, $x^e = 2$ and odd primes, 2 is not relatively prime to $p-1$ so the easy case does not apply.

in $\mathbb{Z}_p^*$ $x \to x^2$ is a 2 to 1 function, i.e. 2 elements of $\mathbb{Z}_p^*$ are going to have the same square root (like they do in real numbers), This also means that only 1/2 the elements of $\mathbb{Z}_p^*$ will have square roots.

x in $\mathbb{Z}_p$ is a quadratic residue (QR) if it has a square root in $\mathbb{Z}_p$

for odd prime # QR in $\mathbb{Z}_p = \frac{(p-1)}{2} + 1 = \frac{p+1}{2}$ [exactly half the number in $\mathbb{Z}_p$ can have square root]
(+1 for 0)

Euler's theorem → x in $\mathbb{Z}_p^*$ is a QR $\Leftrightarrow$ $x^{(p-1)/2} = 1$ in $\mathbb{Z}_p$

$x^{(p-1)/2}$ = Legendre's Symbol of x over p

for odd primes p can be $3 \pmod 4$ or $1 \pmod 4$ [can't be 0 because it'd be divisible by 4; can't be 2 because it'd be even.]

if $p = 3 \pmod 4$

if $c \in \mathbb{Z}_p^*$ is a QR then $\sqrt{c} = c^{(p+1)/4}$ in $\mathbb{Z}_p$

if $p = 1 \bmod 4$, then the square root isn't a cute formula like above but still it can be computed in $O(\log^3 p)$.

Thus: for $\mathbb{Z}_p^*$ given a c we can check if it is a QR using Euler's theorem $c^{(p-1)/2} = 1$ in $\mathbb{Z}_p^*$ and if it exists and $p = 3 \pmod 4$ $\sqrt{c} = c^{(p+1)/4}$ in $\mathbb{Z}_p$

So given $ax^2 + bx + c = 0$ in $\mathbb{Z}_p$ $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ in $\mathbb{Z}_p$

find $(2a)^{-1}$ in $\mathbb{Z}_p$ using ext. euclid algo.
find $\sqrt{b^2 - 4ac}$ using the formulaes above.

ex: $x^2 + 4x + 1 = 0$ in $\mathbb{Z}_{23}$ $\quad x = \dfrac{-4 \pm \sqrt{16-4}}{2 \cdot 1} = \dfrac{-4 \pm \sqrt{12}}{2}$

$x = 5, 14$

23 is $3 \pmod 4$ so $\sqrt{12} = 12^{(23+1)/4} = 12^6$ in $\mathbb{Z}_{23} = 9$.

2^{-1} in $\mathbb{Z}_{23}$, $2 \times -11 + 23 \times 1 = 1 \Rightarrow 2^{-1} = -11 = 12 \pmod{23}$

$-13 \times 12 = +156 \equiv -18 = 5 \bmod 23$

$(-4 \pm 9) \cdot 12 \Rightarrow 5 \times 12 = 14$.

Compute e^{th} root mod composites is hard.

ARITMETIC OPERATIONS MODULO

Representing large numbers, we can use an array of 32/64 bit numbers.

Addition & Substraction: $O(n)$ linear-time in no. of bits of two numbers.

Multiplication: Naive $O(n^2)$ optimized to $O(n^{\log_2^3 1.585})$ Karatsuba

Division with remainder: $O(n^2)$

Exponentiation: $O(n^3)$ Computing x^m where x is a large no. will naively use m multiplications. However this can be very slow if x & m are large. So we use successive squaring algorithm. To beef up the theory we abstract the num system with a finite cyclic group G. (ex. $G = \mathbb{Z}_p^*$)

Give g in G and x compute g^x,

ex. $x = 53$, $x = (110101)_2 \Rightarrow g^{53} = g^{32+16+4+1} = g^{32} \cdot g^{16} \cdot g^4 \cdot g^1$

So we compute $\underline{g}, \underline{g^2}, \underline{g^4}, g^8, \underline{g^{16}}, \underline{g^{32}}$ and then multiply the underlined ones to get g^{53} in 9 mult instead of 53

Input: g in G and $x > 0$; output g^x

write $x = (x_n x_{n-1} \dots x_2 x_1 x_0)_2$ → write x in binary

$\log_2 x$ steps

```
y ← g, z ← 1
for i = 0 to n do:
    if x[i] == 1: z ← z · y      (accumulate product as product of powers of g)
    y ← y²
output z
```

$O(\log x \cdot T_x) = O(\log x \cdot n^2)$

$= O(n^3)$

Intractable Problems

Modular arithmetic is a source of many problems which aren't trivial to solve. For large cases, this fact, is used as the basis of PK crypto. Discrete logarithms is the first one we consider here.

Fix a prime $p>2$ and g in $\mathbb{Z}_p^*$ of order q, consider pow: $x \to g^x$ in $\mathbb{Z}_p$

Consider its inverse $\quad Dlog_g(g^x) = x \quad$ when x in $\{0, \dots q-2\}$

ex: in $\mathbb{Z}_{11} \to Dlog_2(5) = 4$ because $2^4 = 5 \pmod{11}$

$Dlog(\cdot)$ is very difficult in $\mathbb{Z}_p$ if p is a large prime.

A general definition of DLOG.

Let G be a finite cyclic group and g be a generator of G, i.e. $G = \{1, g, g^2, \dots g^{q}\}$ q is the order of G.

then we say DLOG is hard in G if for all efficient algo. A:

$$P_{r\, g \leftarrow G,\, x \leftarrow \mathbb{Z}_q}\left[A(G, q, g, g^x) = x \right] < \text{negligible.}$$

ex. of G where DLOG is hard $\to (\mathbb{Z}_p^*)$ for large primes

$\to$ Elliptic curve groups mod p.

Elliptic curve groups mod p is harder than $(\mathbb{Z}_p^*)$ for large primes

The fact that DLOG is hard on $\mathbb{Z}_p^*$ was used by DH. That same template can be replaced by EC groups, which are harder, so the parameters with EC are smaller and more manageable. TRENDING towards EC

Intractable problems on composites: (a very special case of composites)

$$\mathbb{Z}_{(2)}(n) := \{ N = p \cdot q \text{ where } p, q \text{ are } n\text{-bit primes} \}$$

(product of 2 primes)

2048 ← safe value

Problem 1: factor a random N in $\mathbb{Z}_{(2)}(n)$

$n = 768$ was solved recently

1024 will soon be solved

Problem 2: Given a polynomial $f(x)$ where degree > 1 and a random N in $\mathbb{Z}_2(n)$

find x in $\mathbb{Z}_N$ such that $f(x) = 0$ in $\mathbb{Z}_N$

(RSA uses this for specific $f(x)$)

Public Key Encryption

- Semantic Security of PK encryption ⟹ Eavesdropping security.

b → [Challenger → Adv : P_K
Adv → Challenger : $m_0, m_1 : |m_0| = |m_1|$
Challenger → Adv : $E(P_K, m_b)$] → $b' = EXP(b)$

$Adv^{[A,E]}_{SS} \left| Pr[EXP(0)=1] - Pr[EXP(1)=1] \right| < neg.$

Here note that $E = (G, E, D)$ — G is a randomized algorithm to generate keypair; E is a randomized algorithm to encrypt; D is a deterministic algo to decrypt.

Note that E is randomized which is required because otherwise the attacker has all the knowledge (P_K, m) to generate c and defeat the system with Adv. 1.

Also note: One-time security ⟹ many-time security (CPA) for PK system.

Pub-key Chosen Ciphertext Security ⟹ Security against active attacks

Let $E = (G, E, D)$ be pubkey scheme over (M, C) for $b = 0, 1$ define $EXP(b)$:

b → [
1) Challenger generates $pk, sk \leftarrow G()$;
2) Chal → Adv : pk

CCA phase 1:
3a) Adv → Challenger : $c_i \in C$ — Asks for decryption of ciphertexts of his choice
3b) Challenger → Adv : $m_i \leftarrow D(sk, c_i)$

Challenge:
4a) Adv → Chall : $m_0, m_1 : |m_0| = |m_1|$ — Asks for encryption of messages of his choice
4b) Chall → Adv : $c \leftarrow E(pk, m_b)$

CCA phase 2:
5a) Adv → Chal : $c_i \in C$; $c_i \neq c$ — one more round of ciphertext choice.
5b) Chal → Adv : $m_i \leftarrow D(sk, c_i)$
] → b'

E is CCA secure (IND-CCA) if $\left| Pr[EXP(0)=1] - Pr[EXP(1)=1] \right| < neg.$

Suppose attacker can modify ciphertexts without being detected, active attack, we can show E is not IND-CCA. → b

[Chal → A : pk
A → Chal : (to:alice, 0), (to:alice, 1)
Chal → A : (to:alice, b) = c assume c can be tweaked.
CCA phase 2 — Adv → Chal : (to:charlie, b) = c_i note $c_i \neq c$
Chal → Adv : D(sk, (to:chuk, b)) → b] Adv: 1

TRAPDOOR FUNCTION

A trapdoor function $X \to Y$ is a triple of efficient algorithms (G, F, F^{-1})

- G = randomized algo that outputs (PK, SK)
- $F(PK, \cdot)$: deterministic algo that defines a function $X \to Y$
- $F^{-1}(SK, \cdot)$ = defines a function $Y \to X$ that inverts $F(PK, \cdot)$

$\forall PK, SK$ output by G
$\forall x \in X \quad F^{-1}(SK, F(PK, x)) = x$

trapdoor

A TDF (G, F, F^{-1}) is secure if $F(PK, \cdot)$ is a one way function
Can be evaluated but not inverted without SK.

Challenger: $(PK, SK) \leftarrow G()$, $x \xleftarrow{R} X$
Challenger $\to$ Adv : $PK, y \leftarrow F(PK, x)$
Adv $\to x'$

$Adv_{ow}[A, F] = Pr[x = x'] < \text{negl}$

With Secure TDF, we can build a secure PK system as follows. → SAHAI SG

Given Secure TDF: (G, F, F^{-1}), (E_s, D_s): symmetric AE scheme over (K, M, C), $H: X \to K$ hash funct

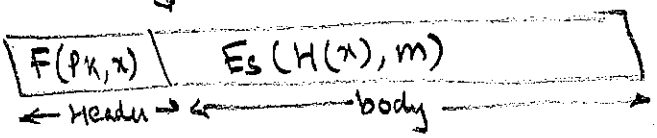
We can construct a PK system $\mathcal{E} = (G, E, D)$ as:

G: Same as TDF G.

$E(PK, m) :=$
$x \xleftarrow{R} X$, $y \leftarrow F(PK, x)$
$K \leftarrow H(x)$, $c \leftarrow E_s(K, m)$
output (y, c)

$D(SK, (y, c)) :=$
$x \leftarrow F^{-1}(SK, y)$
$K \leftarrow H(x)$, $m \leftarrow D_s(K, c)$
output m.

Such a (G, E, D) is CCA secure



ISO standard way of composing PK scheme from TDF & E & H

It is tempting to use a TDF directly as a PK system

Enc $\Rightarrow c = F(PK, m)$ & dec $\Rightarrow m = F^{-1}(SK, m)$

but this would be wrong because it is deterministic & also because the details of practical TDFs will enable attacks.

How do we make a TDF → next.

RSA TRAPDOOR PERMUTATION

RSA is a TDP (Permutation because, it maps $X \to X$). Like a TDF it is defined over a triple (G, F, F^{-1})

G: Choose random primes $p, q \approx 1024$ bits, Set $N = p \times q$ → modulus.

Choose integers e, d $\quad e \cdot d = 1 \pmod{\phi(N)}$ $\quad \phi(N) = N - p - q + 1$

output $PK = (N, e)$ $\quad SK = (N, d)$

e = encryption exponent, d = decryption exponent

e, d are relatively prime to $\phi(N)$ & e & d are inverses of each other

$$F(PK, x) : \mathbb{Z}_N^* \to \mathbb{Z}_N^* : RSA(x) = x^e \bmod N$$

$$F^{-1}(SK, y) : y \bmod N \to (RSA(x))^d \Rightarrow (x^e)^d \bmod N = x^{ed} \bmod N$$

but $ed = 1 \bmod \phi(N)$, that is $e \cdot d = K\phi(N) + 1$

$$x^{ed} \bmod N = x^{K\phi(N)+1} \bmod N = (x^{\phi(N)})^K \cdot x = x.$$

↓ Euler's theorem = 1.

RSA is a secure system because decryption requires 'd' and that requires computing $\phi(N)$ which in turn requires factorizing n into p, q which is hard.

RSA assumption: for all efficient algo A: where $p, q \xleftarrow{R}$ n bit primes

$$Pr[A(N, e, y) = y^{1/e}] < \text{negligible}. \quad N \leftarrow pq,\ y \xleftarrow{R} \mathbb{Z}_N^*$$

Note that RSA is a TDF & not an encryption scheme. It should not be used as the encryption system itself. We should plug it into ISO scheme we saw earlier.

$G()$: generate $PK = (N, e)$ $\quad SK = (N, d)$

$E(PK, M)$: choose random x in $\mathbb{Z}_n$, $y \leftarrow RSA(x) = x^e$, $K \leftarrow H(x)$, output $(y, E_S(K, m))$

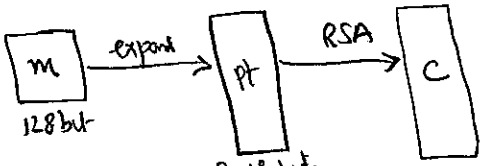
$D(SK, (y, C))$: output $D_S(H(RSA^{-1}(y)), C)$

Using RSA by itself can be broken in less than exhaustive search by MITM (Brute force) because for $C = K^e$, $K = K_1 \cdot K_2$ prob ≈ 20% $\quad C/K_1^e = K_2^e$ → vary K_1, compute all possible values

PKCS

Never use textbook RSA. ISO scheme can be used, but in practise PKCS is used.

Generally, RSA is used to encrypt secret keys, say 128 bit AES key: enc(k).

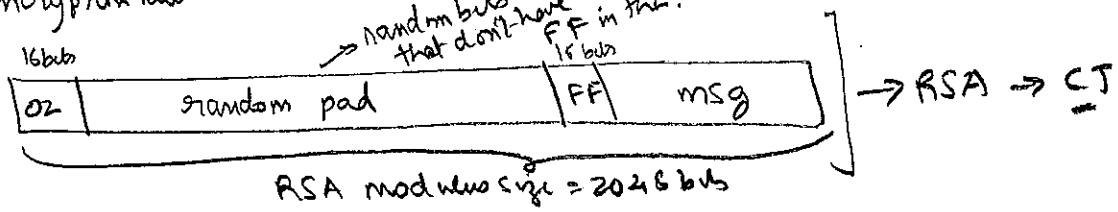


Questions one - how should we expand the short message to the input PT for RSA encryption? 2 standard ways of doing it

modes - signature

encryption mode

PKCS 1 mode 2



PCKS1 v1.5

We take the message, pref ix with FF, set the MSB 16 bits to 02 to identify the scheme we're using and fill the rest with random bits that don't include FF.

Widely used, eg. in HTTPS → Was broken by Bleichenbacher 1998 using the fact that the protocol responded differently when the $msb_{16}(PT) \neq 02$.

This was used as the oracle to expose the PT, using a chosen-ciphertext attack.

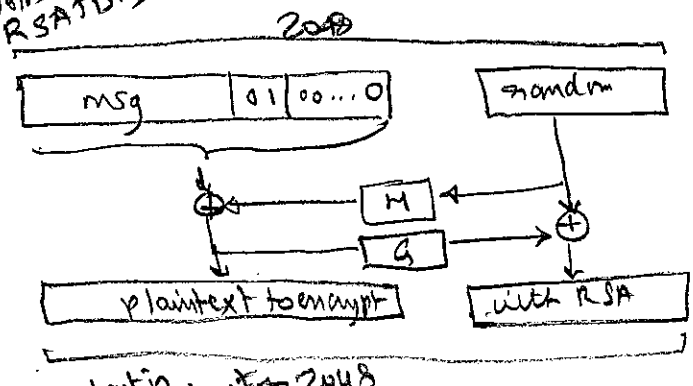
Bleich. attack asks the WS to decrypt- $r \in \mathbb{Z}_N$ $c' \leftarrow r^e c = (r \cdot PKCS_1(m))^e$

and detects if the most significant bits are 02 or not.

HTTPS was modified (RFC 5246). → Instead of responding with error if PT does not start with 02, we assume all is well and use a random R as the message.

Protocol will fail later because R will not be the right m → which is BTW the pre-master-secret.

OAEP (only works with RSATDF)



Optimal Asymmetric Encryption padding

H & G are random oracles → in practise SHA-256

OAEP+, SAEP → simple

SAEP → Used if e = 3 → encryption exponent

OAEP+ → extends OAEP to general TDP (and not just RSA)

Uses W	replaces padding with W(M,r)
another Oracle.	

Most used in PKCS1 1.5 in HTTPS

OAEP is used 2nd Most.