Introduction to Modern Cryptography

Benny Chor

Iterated Ciphers
Message Authentication Codes
Cryptographic Hash Functions

Lecture 4

Tel-Aviv University

November 10, 2008

Strengthening a Given Block Cipher

A block cipher, like DES, is initially considered secure. It gains wide usage, but as time passes, it is realized that

- due to technological advances exhaustive search becomes feasible

  and/or
- due to algorithmic advances, some specialized attack becomes feasible.

As the person in charge of security in a large company, what do you do?

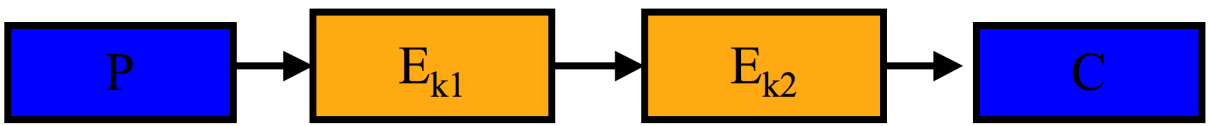
1. Switch to a fancy new block cipher (how much would that cost? *e.g.* suppose the old cipher is physically embedded in all bank ATM machines).
2. Find ways to use the existing cipher with enhanced security (great idea, but how?).

Iterated Ciphers

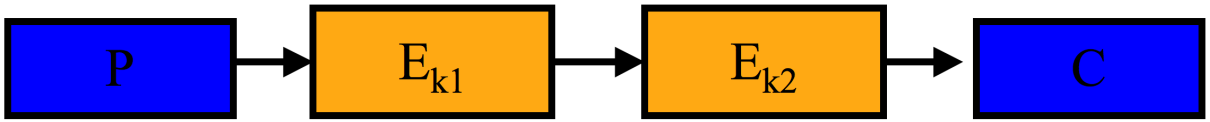
Suppose E_k is a good encryption function, mapping $\{0,1\}^n$ to itself. It is not possible to distinguish efficiently (poly time in n) between pairs $(x_i, E_k(x_i))$ and random pairs $(x_i, y_i) \in \{0,1\}^n$. For simplicity, further assume that the key space is also $\{0,1\}^n$.

New technological advances made exhaustive search over the key space (time $O(2^n)$) a possibility. Given a few (but more than one) pairs $(x_i, E_k(x_i))$, we can go over all keys and find the one that matches all pairs. This takes $O(2^n)$ time and $O(1)$ pairs have to be stored.

Idea: Instead of using a single key, k, use two keys, k_1, k_2. To encrypt x, compute $(E_{k_2}(E_{k_1}(x)))$. This should double the key space, thus hopefully squaring the running time of an exhaustive attack, from $O(2^n)$ to $O(2^{2n})$, giving us another decade or two of tight sleep :-)



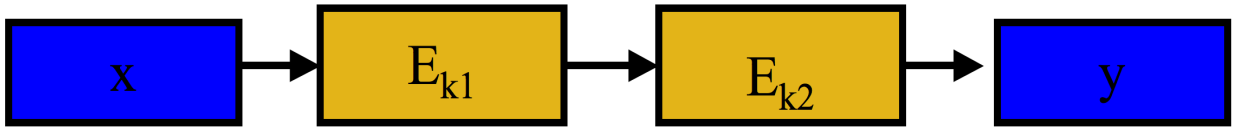
Double Cipher



This sounds like a good idea, but there are a few caveats:

- If encryption is closed under composition, namely for all k_1, k_2 there is k_3 such that $(E_{k_2}(E_{k_1}(x))) = E_{k_3}(x)$, then we gain nothing. Could just exhaustively search for such k_3.
- Substitution ciphers certainly posses this property. In fact they constitute a permutation group, hence closed under composition.
- It was suspected that DES also acts as a group (or is almost one) under composition. Quite a lot of research was devoted to this question, and it was refuted only in 1992.
- Notice that if an encryption scheme is closed under composition, then not only double iteration, but any fixed number of iterations are useless.

Meet in the Middle Attack



Goal: Recover the two keys k_1, k_2. We start with one pair of known plaintext and ciphertext (x, y), such that $E_{k_2}(E_{k_1})(x) = y$.

1. For each key k_1, compute $z_{k_1} = E_{k_1}(x)$. Store the list of pairs $L_1 = \{(z_{k_1}, k_1)\}$, sorted by z_{k_1}.
2. For each key k_2, compute $t_{k_2} = D_{k_2}(y)$. Store the list of pairs $L_2 = \{(t_{k_2}, k_2)\}$, sorted by t_{k_2}.
3. Each time we have $z_{k'} = t_{k''}$, the pair of keys (k', k'') are a potential solution to $E_{k''}(E_{k'}((x)) = y$.
4. We expect $O(2^n)$ potential solutions. This is much better than $O(2^{2n})$ (all pairs), but not good enough.
5. To wrap things up, do the same process for one or two more pairs $(x_1, y_1), (x_2, y_2)$. Intersecting the set of solutions, we will be left (with high probability) with the single "correct" solution (k_1, k_2).

Intuition for Analyzing Meet in the Middle

Question: Given a key k_1 and $x, y \in \{0,1\}^n$, denote $z = E_{k_1}(x)$. Is there a key k_2 such that $z = D_{k_2}(y)$?

Actually the answer could be negative. $z = D_{k_2}(y)$ is equivalent to $y = E_{k_2}(z)$. If we fix z and look at $E_{k_2}(z)$ as a function of k_2, then there are no guarantees on this function.

We expect this mapping to be random or close to random, but there is no reason to expect it to be a permutation of $\{0,1\}^n$, so it need not be onto. (In fact, if the key length is shorter than n, then this mapping will certainly not be onto.)

Assume the length of the key is ℓ, (not necessarily equal to n). Then for any fixed x, y, $E_{k_1}(x)$ and $D_{k_2}(y)$ should behave as two random independent functions, each mapping $\{0,1\}^\ell$ into a subset of $\{0,1\}^n$ (these are functions of k_1 or k_2).

Intuition for Analyzing Meet in the Middle (cont.)

Each function ($E_{k_1}(x)$ and $D_{k_2}(y)$) maps 2^ℓ keys to 2^n elements of $\{0,1\}^n$. Thus the probability that $E_{k_1}(x)=D_{k_2}(y)$ is $1/2^n$. The number of pairs k_1, k_2 is $2^{2\ell}$. Thus the expected number of pairs that achieve $E_{k_1}(x)=D_{k_2}(y)$ is exactly $2^{2\ell-n}$. This is the number of matches we expect to find in L_1, L_2 for a single plaintext-ciphertext pair (x, y).

What is the expected number of keys k', k'' that will produce mathches for two plaintext-ciphertext pairs $(x_1, y_1), (x_2, y_2)$? Exactly $2^{2\ell-2n}$, which (for $\ell \leq n$) is at most 1.

If $(x_1, y_1), (x_2, y_2)$ are correct pairs, satisifying $E_{k_1}(x_1)=D_{k_2}(y_1)$ and $E_{k_1}(x_2)=D_{k_2}(y_2)$, then except the pair of keys k_1, k_2, we expect to find no other pairs k', k''. If any is found, a third (x_3, y_3) will almost surely reduce the number of spurious keys to zero.

Meet in the Middle Attack: Performance

Time: $O(n2^n)$ (sorting a 2^n long list).
Space: Must store $O(2^n)$ pairs.

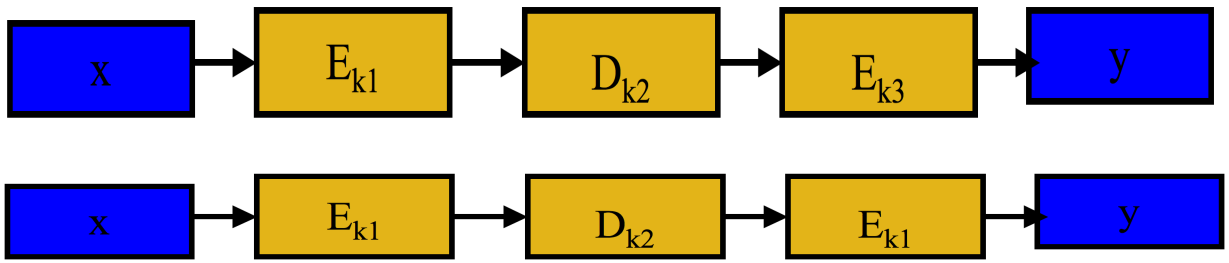
Compare this to exhaustive search over the space of a single key:

Time: $O(2^n)$.
Space: Must store just $O(1)$ pairs.

Conclusion: Double cipher (two iterations) does not add much to security (though it adds some, esp. if fast memory is expensive).

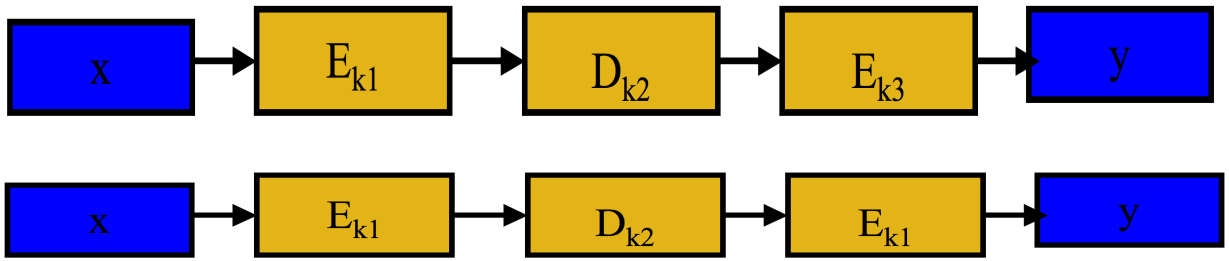
Let us then move to triple cipher (three iterations).

Triple Cipher



- The first version utilizes three independent keys k_1, k_2, k_3. The second, just two – k_1, k_2.
- The "middle box" employs decryption rather than encryption. This results in backward compatibility: Taking $k_1, k_2, k_3{=}k$, this is identical to a single encryption using k.
- When the key is unknown, decryption (should) also be a pseudo random function, so using D_{k_2} does not reduce security.

Triple Cipher / Triple DES



- Both versions are susceptible to a meet in the middle attack, albeit one that uses a list of size 2^{2n} (where n is length of keys). This is the best attack known for version (1).
- For version (2), there is a $O(n2^n)$ attack using $O(2^n)$ chosen input/output pairs. This substantially reduces the practical risk of such attack. Triple DES, using version (1), has key size $3 \cdot 56$ bits, and best attack known is meet in the middle, with complexity $2^{2 \cdot 56} = 2^{112}$. It is considered secure, and was endorsed by a standard of NIST. However, AES is pushing it out of the market.

From Encryption to Authentication

But First, Some Changes in Grading Procedures

From: benny@cs.tau.ac.il
Subject: Exam in Crypto Course - Clarification
Date: May 35, 2010 6:33:23 AM GMT+02:00
To: 0368-3049-01@listserv.tau.ac.il
Reply-To: benny@cs.tau.ac.il

To all students in the course,

Due to the strike, I decided to change the regular exam procedure. You will be able to take a a normal exam, or opt not to take one. In the later case, your final grade in the course will be your average courses' grade (up to the exam date) plus 10 points (if this exceeds 100, the grade will be just 100). If you decide to take the exam, which I'd like to encourage you to do, your final grade will be the maximum of the exam grade and your average plus 10 (as above).

Please note that this is a numeric grade and not a pass/fail one, which was banished by the university senate. In addition, this arrangement has been explicitly approved by the Dean and the Rector.

Sincerely,

Benny Chor

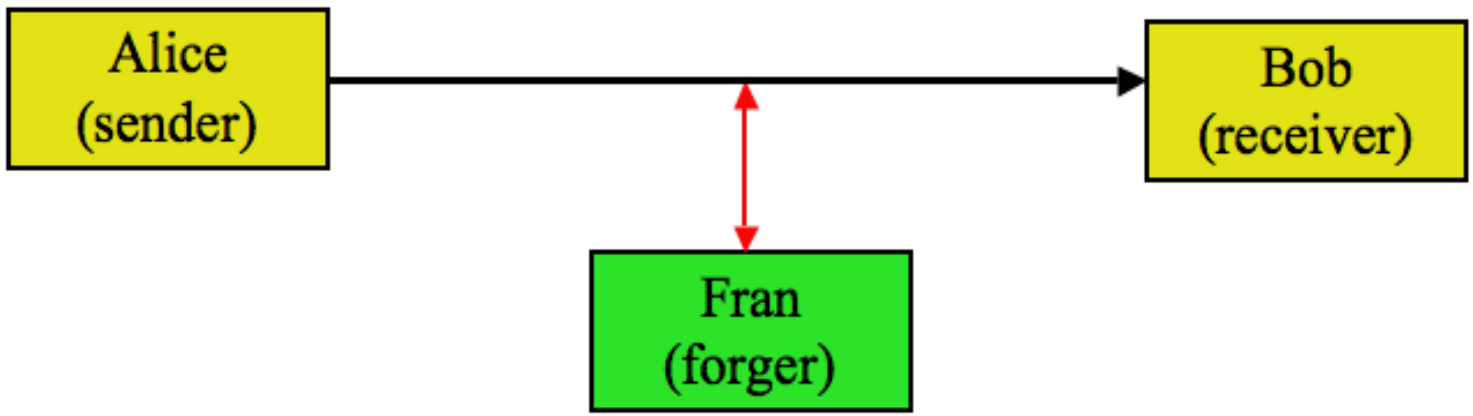
School of Computer Science, Tel-Aviv Univ.
www.cs.tau.ac.il/∼bchor

- -

- Would you believe this message?
- Do you really think it was originated from `benny@cs.tau.ac.il` (your lecturer)? How can you tell?
- How many of you can actually forge such message and distribute it to the course mailing list?

Authentication – Goal

Ensure integrity of messages, even in presence of an active adversary who hears previous genuine messages (in a worst case scenario, these could possibly include messages she chose), and then sends own forged message(s). Bob (receiver) should be able to tell genuine messages from forged ones.



Important Remark: Authentication is orthogonal to secrecy, yet systems often required to provide both. However, the two are typically handled separately, then combined to one message. Secrecy alone usually does not guarantee integrity.

Authentication: Notation and Definition

- Authentication algorithm – A
- Verification algorithm – V ("accept" / "reject")
- Authentication key – k
- k is a secret key, shared among receiver and sender
- Message space (usually binary strings)
- Every message between Alice and Bob is a pair $(m, A_k(m))$
- $A_k(m)$ is called the authentication tag of m

Authentication: Notation and Definition (cont.)

- Consistency requirement: $V_k(m, A_k(m)) =$ "accept"
- The authentication algorithm is called MAC (Message Authentication Code)
- $A_k(m)$ is frequently denoted $MAC_k(m)$
- Verification is done by executing authentication algorithm on m and k, and comparing with the value received, $MAC_k(m)$

MAC Functions: Requirements and Properties

- Security requirement – any computationally bounded adversary cannot construct a new legal pair $(m, MAC_k(m))$, even after seeing n legal pairs $(m_i, MAC_k(m_i))$ $(i = 1, 2, \ldots, n)$, except with negligible probability.
- Output should be as short as possible
- Thus the MAC function is not 1-to-1

We will say that the adversary succeeded even if the message Fran forged is "meaningless". The reason is that it is hard to predict what has and what does not have a meaning in an unknown context, and how will Bob, the reciever, react to such successful forgery.

Adversarial Model

- Data and Information Available to Adversary:
  - The MAC algorithm (but not the secret key)
  - Either Known plaintext and authentication tag pairs
  - Or Chosen plaintext and authentication tag pairs
- Note: chosen MAC is typically unrealistic
- Adversary goal: Given n legal pairs $(m_1, MAC_k(m_1)) \ldots, (m_n, MAC_k(m_n))$, find a new legal pair $(m, MAC_k(m))$

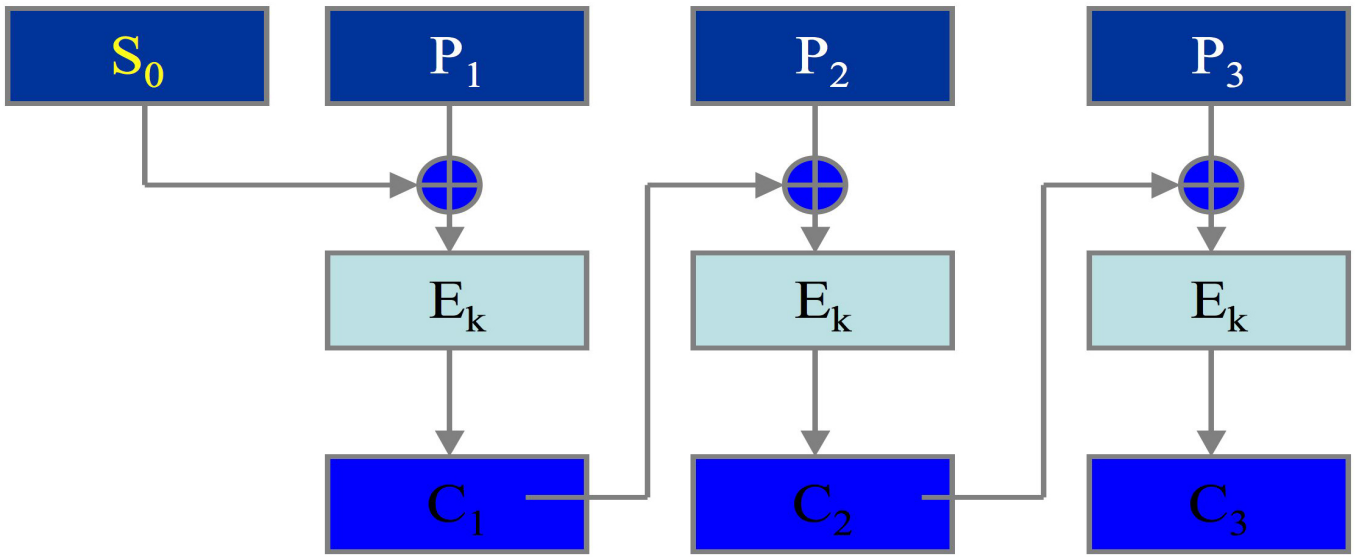
Efficiency

- Adversary Goal: Given n legal pairs $(m_1, MAC_k(m_1)) \ldots, (m_n, MAC_k(m_n))$, find a new legal pair $(m, MAC_k(m))$ efficiently and with non negligible probability.
- If n is large enough, then n pairs $(m_i, MAC_k(m_i))$ will determine the key k uniquely (with high prob.). Thus a non-deterministic machine can guess k and verify it. But doing this deterministically should be computationally hard.

MACs Used in Practice

We will describe a MAC based on CBC Mode Encryption, and a MAC based on cryptographic hash functions.

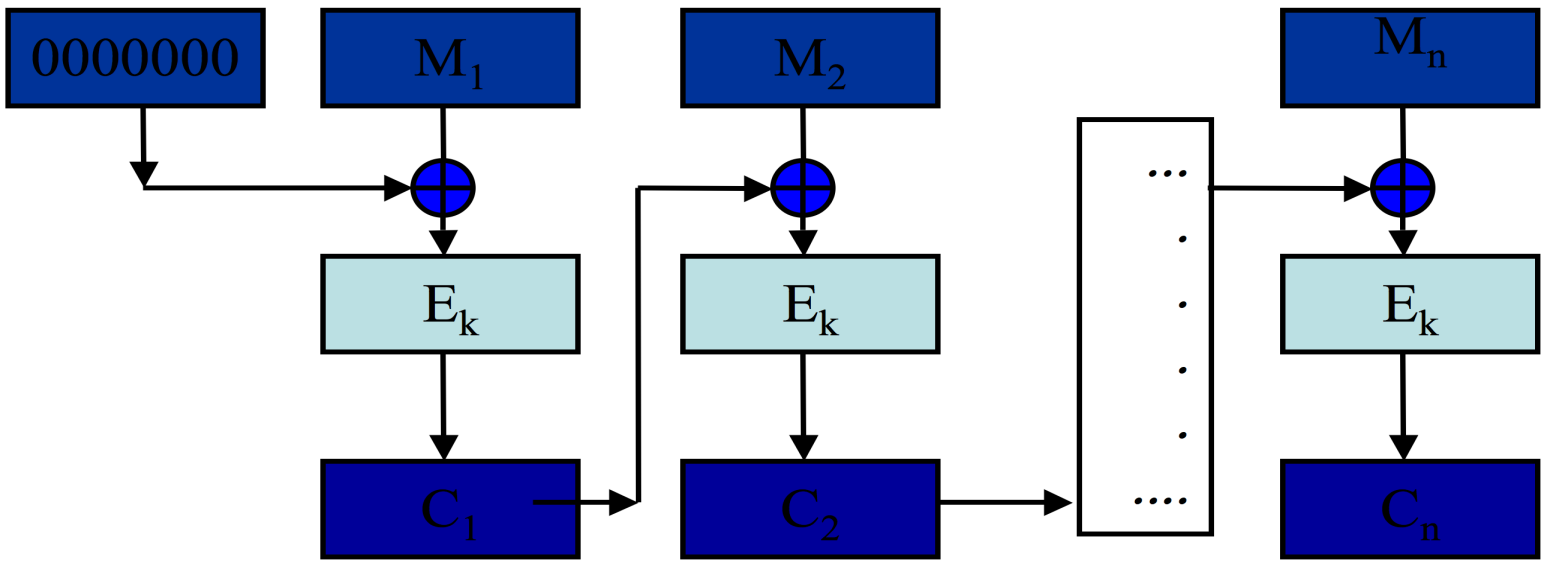
Reminder: CBC Mode Encryption.



In CBC mode (Cipher Block Chaining), previous ciphertext is XORed with current plaintext before encrypting current block. The initialization vector S_0 is used as a seed for the process. It can be transmitted "openly".

CBC Mode MACs

- Start with the all zero seed.
- Given a message consisting of n blocks, $M_1, M_2, \ldots, M_n$, apply CBC mode encryption (using the secret key k).



- Produce n "cipertext" blocks, $C_1, C_2, \ldots, C_n$.
- Discard first $n-1$ blocks.
- Send $M_1, M_2, \ldots, M_n$ and the authentication tag $MAC_k(M) = C_n$.

Security of Fixed Length CBC MAC [BKR, 2000]

- Theorem: If E_k is a pseudo random function, then the fixed lenght CBC MAC is resilient to forgery when authenticating messages of the same length, n.

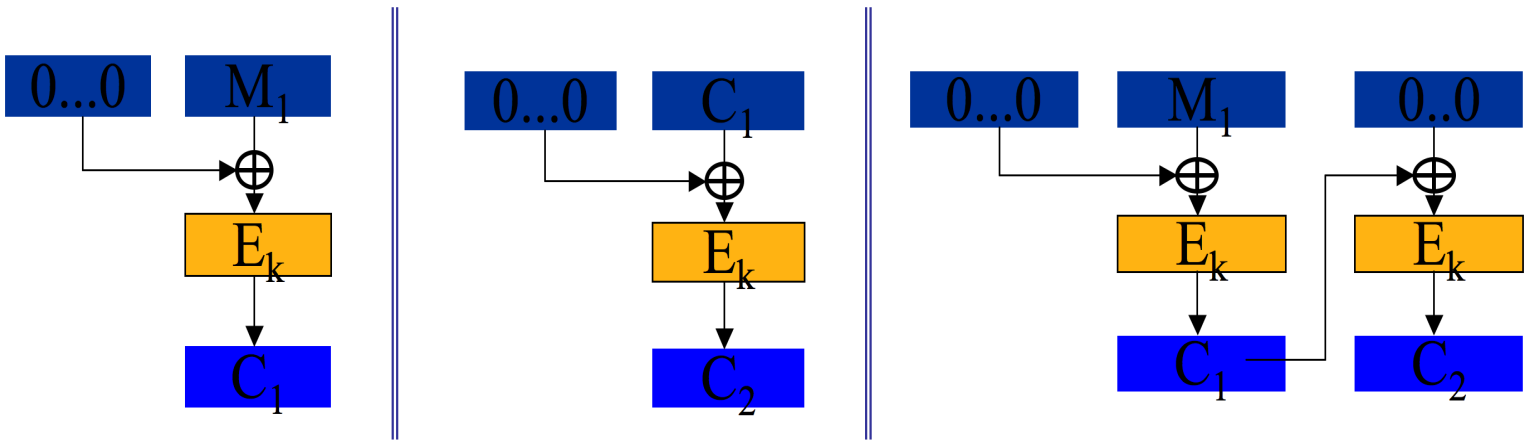
- Proof outline: Assume CBC MAC can be forged efficiently. Transform the forging algorithm into an algorithm distinguishing E_k from a random function efficiently.

- Warning: Construction is not secure if messages are of varying lengths, namely number of blocks varies among messages.
- Construction is secure if message space is prefix free. Fixed length messages constitute one such space.
- Comment: There are reasonably simple ways to extend security to variable lengths messages.

Insecurity of Variable Length CBC MAC

Here is a simple, chosen plaintext example of forgery:

- Get $C_1 = CBC - MAC_k(M_1) = E_k(\vec{0} \bigoplus M_1)$
- Ask for MAC of C_1, i.e., $C_2 = CBC - MAC_k(C_1) = E_k(\vec{0} \bigoplus C_1)$
- Observe that $E_k(C_1 \bigoplus \vec{0}) = E_k(E_k(\vec{0} \bigoplus M_1) \bigoplus \vec{0}) =$ $CBC - MAC_k(M_1 \circ \vec{0})$ (where $\circ$ denotes concatenation)



- One can efficiently design, for every n, two messages, one with 1 block, the other with $n+1$ blocks, that have the same $MAC_k(\cdot)$.

Combining Authentication and Secrecy

- Given a message consisting of n plaintext blocks, $M_1, M_2, \ldots, M_n$, apply CBC (using the secret key k_1) to produce $MAC_{k_1}(M)$.
- Using a different, independently chosen key, k_2, produce n ciphertext blocks $C_1, C_2, \ldots, C_n$ under k_2.
- Send $C_1, C_2, \ldots, C_n$ and the authentication tag $MAC_{k_1}(M)$.

CBC-MAC for Variable Length Messages

- Solution 1: The first block of the message is set to be its length. Namely, to authenticate $M_1, \ldots, M_n$ apply CBC-MAC to $(n, M_1, \ldots, M_n)$. Works since now message space is prefix-free. Drawback: The message length, n, must be known in advance.
- "Solution 2": apply CBC-MAC to $(M_1, \ldots, M_n, n)$ Message length does not have to be known is advance. Looks good, but this scheme was broken (see, M. Bellare, J. Kilian, P. Rogaway, The Security of Cipher Block Chaining, 1984)
- Solution 3: (recommended) Use a second key secret k_2. Compute
  $MAC_{k_1,k_2}(M_1, \ldots, M_n) = E_{k_2}(MAC_{k_1}(M_1, \ldots, M_n))$
  Essentially the same overhead as CBC-MAC.

Hash Functions: Reminder

- Hash functions map large domains $\mathcal{X}$ to smaller ranges $\mathcal{Y}$.
- Example: $h : \{0, 1, \ldots, p^2\} \mapsto \{0, 1, \ldots, p-1\}$, defined by $h(x) = a \cdot x + b \mod p$.
- Hash tables are extensively used for searching.
- If the range is smaller than the domain, there could be collisions ($x \neq y$ with $h(x) = h(y)$). For example, in the hash function above, if $x_1 = x_2 (\mod p)$ then $h(x_1) = h(x_2)$.
- A good hash function should create few collisions for most subsets of the domain ("few" is relative to size of subset).
- In data structures, collisions are resolved by several possible means – chaining, double hashing, etc.
- Hash functions, including cryptographic ones, have no secret key.

Security Requirements from Cryptographic Hash Functions

1. Pre-image resistance: for any y, it is hard to find x such that $h(x) = y$.
2. Weak collision resistance: for any $x_1 \in \mathcal{X}$, it is hard to find $x_2 \neq x_1$ such that $h(x_1) = h(x_2)$. (This requirement is also known as "universal one-way hash", or "second preimage resistance").
3. Strong collision resistance, aka "claw freeness": it is hard to find any pair $x_1, x_2 \in \mathcal{X}$ such that $h(x_1) = h(x_2)$.

Under reasonable assumptions, strong collision resistance implies the two other properties. Thus in general it will be harder to satisfy.

The Birthday "Paradox"

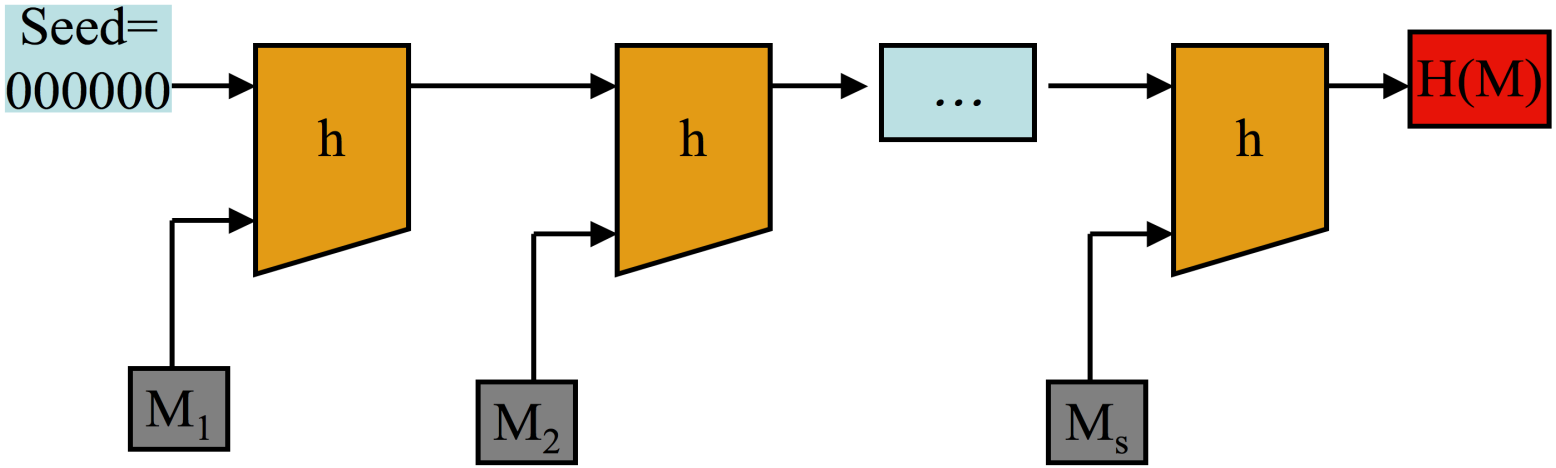
- If 23 people are chosen at random, the probability that two of them have the same birth-day (day & month only) is greater than 0.5.
- Compare to: the prob. that one or more of them has the same birthday as Claude Shannon is $\approx 23/365$ (more precisely, $1 - \left(1 - \frac{1}{365}\right)^{23}$).
- More generally, let $h : \mathcal{X} \mapsto \mathcal{Y}$ be a random mapping. If we chose $1.17|\mathcal{Y}|^{1/2}$ elements of $\mathcal{X}$ at random, the probability that two of them are mapped to the same image is greater than 0.5.
- This implies that strong collision resistance is easier to violate than weak collision resistance.
- Suppose $h : \mathcal{X} \mapsto \mathcal{Y}$ and $|\mathcal{Y}| = 2^n$. Then picking approximately $2^{n/2}$ elements of $\mathcal{X}$ at random, we will find, with high probability, $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$. On the other hand, to find an x such that $h(x) = h(0)$, we need approximately 2^n random elements of $\mathcal{X}$.

Cryptographc Hash Functions

Hash functions $h : \{0,1\}^n \mapsto \{0,1\}^m$, satisfying:

- Strong collision resistance.
- Very fast to compute.
- Recall: no secret key.
- $h(x)$ is often called the digest of x.
- In "real life", input block length is usually $n = 512$ bits ($|\mathcal{X}| = 2^{512}$).
- Output length is at least $m = 160$ bits (to foil birthday attacks).

Extending to Variable Length Messages



- Suppose $h : \{0,1\}^{512} \mapsto \{0,1\}^{160}$.
- The input message is $M = M_1 M_2 \ldots M_s$. The length of each M_i is $512 - 160 = 352$ (what if 352 does not divide $|M|$?).
- Define $y_0 = \text{seed} = 0^{160}$, $y_i = h(y_{i-1}, m_i)$; $y_{s+1} = h(y_s, m_{s+1})$, $h(M) = y_{s+1}$.
- Is this secure? What about input messages of different lengths?
- Can show that collisions in H imply collisions in h.

Real World Cryptographc Hash Functions

- MD family ("message digest")
  - MD-2
  - MD-4 (full description in Stinson's book)
  - MD-5
  - MD-5 hashes to 128 bit strings. This relatively small size was exploited to find collisions, and MD-5 is now considered broken. See `http://en.wikipedia.org/wiki/MD5#Vulnerability`
- SHA and SHA-1 (secure hash standard, 160 bits) (`www.itl.nist.gov/fipspubs/fip180-1.htm`)
- (Apparently for SHA-0, just 2^{39} applications are now required to find a collision, and 2^{63} are required for SHA-1.)
- RIPE-MD
- SHA-256, 384 and 512 (proposed standards, longer digests)

Real World Cryptographc Hash Functions, cont.

Interestingly, these very days (fall 2009), NIST is conducting a public competition to develop a new cryptographic hash algorithm.

The competition is NIST's response to recent advances in the cryptanalysis of hash functions.

The new hash algorithm will be called SHA-3.

Criteria are speed and some "proof" of security.

See `http://csrc.nist.gov/groups/ST/hash/sha-3/index.html`

Using Cryptographic Hash Function to Build MACs

- Hash functions are not keyed. MAC_k does use a key.
- Best attack should not succeed with probability greater than $\max\left(2^{-|k|}, 2^{-|MAC(\cdot)|}\right)$.
- Idea: Combine message and the secret key, then hash them with a collision resistant hash function.
- Nice idea, but how? The devil is in the details.
- Two possible implementations:
  1. $MAC_k(M) = h(k, M)$.
  2. $MAC_k(M) = h(M, k)$.
- Both turn out to be insecure.

HMAC

- Proposed in 1996 by [BCK].
- Receives as input a message M, a key k and a hash function, h.
- Outputs a MAC by:
  $HMAC_k(M, h) = h(k \bigoplus \texttt{opad}, h(k \bigoplus \texttt{ipad}, M))$.
- The two strings opad and ipad are 64 byte long fixed strings.
- k is 64 byte long (if shorter, append 0s to get 64 bytes).
- Theorem [BCK]: HMAC can be forged if and only if the underlying hash function is broken (collisions found).
- HMAC is extensively used (*e.g.* SSL, IPSec).