Outline

- MAC - Message Authentication Code
- MAC usage and definition
- CBC mode MAC
- Hash based MAC
- Weak keyed CRHF
- Nested MAC (NMAC)
- Proof of security (reduction)
- Hash based MAC (HMAC)
- Shared Key Mutual Authentication
- Combining Encryption and MAC
Message Authentication Problem

- Detect changes by adversary to message
  - Ancient solution: sign and seal
    - More on this in next subject…
  - Even more ancient: break brick to message part and authenticator part (‘tag’)
    - Send tag to recipient securely (in advance)
    - Send message over insecure channel (later)
    - Recipient validates message using tag
  - How to do this electronically?
- First idea: tag is $h(m)$ where $h$ is CRHF
  - Send tag securely…
Message Authentication by CRHF

- Alice sends (securely) the hash of the message to Bob, then (insecurely) the message itself
- Bob computes hash of received message, compared to received hash (tag)
- Question: why send securely hash, not message?
  - Do we need secure channel?
Detection of Message Modification

- Can we simply send the hash with the message?
  - A: No; Eve can then change message and compute new hash!
Message Authentication Code (MAC)

- **Message Authentication Code – MAC**
  - $\text{Valid}_k(\text{Tag} , m)=\text{True} \iff \text{Tag}=\text{MAC}_k(m)$

- **Use shared key $k$ to authenticate messages**

Message Authentication Code (MAC)

- **Message Authentication Code – MAC**
  - $\text{Valid}_k(\text{Tag} , m)=\text{True} \iff \text{Tag}=\text{MAC}_k(m)$

- **Use shared key $k$ to authenticate messages**
Defining MAC: Syntax and usage

- A Message Authentication Code is an efficient function $MAC_k(m): \{0,1\}^l \times \{0,1\}^* \rightarrow \{0,1\}^l$.
  - Also called VIL-MAC (Variable Input Length)
  - $n \times l$ FIL-MAC: $\{0,1\}^l \times \{0,1\}^{ln} \rightarrow \{0,1\}^l$

- To authenticate $m$, send $\langle m, MAC_k(m) \rangle$

- Upon receiving $\langle m, a \rangle$, verify that $a = MAC_k(m)$

- We could also define a Validation function, i.e. $Valid_k(m,a) = TRUE$ iff $a = MAC_k(m)$
  - This is more general, allows randomized MAC
  - But for all practical (and 99% of proposed) MAC, validation is simple compare as above
Simple MAC functions

- Examples of Fixed-Input Length (FIL) MAC: \( \{0,1\}^{16} \times \{0,1\}^{32} \rightarrow \{0,1\}^{16} \)
- Encode input strings (4 chars) by their ASCII encoding
  - \( MAC_k(x) = \text{int}(32768 \times \text{fraction}(x \cdot a + k \cdot b)), \) where \( 0 < a, b < 1 \)
  - \( MAC_k(x) = \text{int}(\lfloor x \parallel k \rfloor \mod 2^{32}) \)
  - \( MAC_k(x) = x \cdot (32567 + k) \mod 32767 \)
  - \( MAC_k(x) = (x[0...15] + k) \oplus (x[16...31] + k) \)
Known Message Attack on MAC

A Message Authentication Code is an efficient function $MAC_k(m): \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^n$.

**Known Message Attack (KMA) Game** on $MAC_k(m)$:

- Pick random key $k \in_R \{0,1\}^n$;
- $i \leftarrow 1$; Repeat until adversary aborts:
  - Adversary specifies $l$, length of message
  - Let $m_i \in_R \{0,1\}^l$
  - Adversary receives $<m_i, MAC_k(m_i)>$
  - Adversary aborts or sets $i \leftarrow i+1$ and repeats;
- Adversary outputs $<m, a>$
- Adversary **wins** if $a=MAC_k(m)$ and $m \neq m_i$ for some $I$

$MAC_k(m)$ is KMA **secure** if no adversary can **efficiently** and with **significant** probability win KMA Game on $MAC_k(m)$.
Security Definitions of MAC

- $MAC_k(m)$ is $(t,\varepsilon)$-KMA-secure if no adversary can win KMA Game on $MAC_k(m)$ with probability greater than $\varepsilon$ using up to $t$ steps (time units).

- $MAC_k(m)$ is asymptotically (polytime) KMA-secure if for every PPT adversary, $\text{Prob}(\text{win}) \in \text{Negl}$

- $MAC_k(m)$ is feasible-KMA secure if it is $(t,\varepsilon)$-KMA-secure, for `feasible` $(t,\varepsilon)$
  - Feasible: considering adversary resources and using Moore’s law (speed doubles / cost halves every 18 months)
Chosen Message Attack (CMA) on MAC

- **Chosen Message Attack** (CMA) on $MAC_k(m)$:
  - Pick random key $k \in \{0,1\}^n$
  - $i \leftarrow 1$
  - Repeat until adversary aborts:
    - Adversary specifies $m_i \in \{0,1\}^*$
    - Adversary receives $<m_i, MAC_k(m_i)>$
    - Adversary aborts or sets $i \leftarrow i+1$ and repeats;
  - Adversary outputs $<m, a>$
  - Adversary **wins** if $a=MAC_k(m)$ and $m \neq m_i$ for all $I$
- $MAC_k(m)$ is CMA-secure if no adversary can efficiently and with significant probability win CMA
MAC authenticates messages

- MAC allows two or more mutually trusting parties to authenticate all messages sent between them.

Alice
Key $k$

$$m, \text{MAC}_k(m)$$

Bob
Key $k$

Only Alice and me know $k$. So one of us sent $m$.

If I wouldn’t send $m$, then Alice must have sent it

Eve
MAC authenticates messages

- MAC allows two or more mutually trusting parties to authenticate all messages sent between them.

Alice

Charlie

Key $k$

Don

Key $k$

Bob

Eve

Key $k$

Only Alice, Charlie, Don and me know $k$. So one of us sent $m$. $m, \text{MAC}_k(m)$
MAC Use Secret Key

- Random or pseudo-random
- Do not reveal to adversary
- Do not use for anything except MAC

Easy to demonstrate:

- $MAC_k, \ Encryption_k$ exposing key $k$
- $MAC_k, \ Encryption_k$ allowing attack
- Example: Let $E, \ MAC$ be secure…
- Forge: Let $E'(m) = E_k(m) || MAC_k(m+1)$
- Expose $m$: Let $MAC'(m) = MAC_k(m) || m$
Limitations of MAC

- $MAC_k(m)$ may expose information about $m$!
  - Example: Let $MAC$ be any secure MAC. Define $MAC'_k(m) = LSb(m) || MAC_k(m)$, where $LSb$ is least significant bit.
- MAC only shows a key-holder computed it
- Could be any key holder (e.g. recipient)...
  - Specify sender, recipient in message
- Could be re-transmission...
  - Add time/counter/random challenge to identify
MAC authenticates messages

- MAC allows two parties sharing random key $k$ to authenticate messages sent between them.
- **Claim 1**: Choose key $k$ randomly and share it between two parties $\{A,B\}$. Assume $A$ and $B$ use key $k$ only to compute $MAC_k(m)$, for different messages $m$. If $B$ receives $m',a'$ s.t. $a' = MAC_k(m')$, then either $A$ or $B$ itself previously computed $MAC_k(m')$.
- **Note**: holds also for pseudorandom $k$
  - Output of PRG or $k=PRP_k'(const)$ where $k' \in_R \{0,1\}^n$
  - Otherwise: use this to distinguish $k$ from random
Constructing MAC

- How to select/design good (secure) MAC?
- Cryptoanalyze candidates against MAC def
  - Problem: requires much effort to be trusted
- Tolerant construction from multiple candidate MAC: *Parallel construction*
  - Exercise: show that $MAC_{k,k'}(m)=f_k(m)||f_{k'}(m)$ is a secure MAC, provided *either* $f$ *or* $f'$ is secure MAC (KMA and CMA)
- Provable-secure constructions from other crypto primitives (which?)
Constructing MAC from...

- Hash functions: see later...
- Pseudo-Random Permutation (block cipher)?
  - Problem: output is same length as input...
- Pseudo-Random Functions?
  - Input can be longer than output...
  - Output can be fixed-length (for MAC)
  - Implement using `modes` of deterministic block ciphers (DES, AES, ...)
  - Fixed Input Length (FIL) or Variable Input Length (VIL)
Fixed Input Length PRF

An $n \times l$ FIL-PRF: collection of efficient functions $\{f_k : \{0,1\}^n \rightarrow \{0,1\}^l \}$, such that no adversary can efficiently distinguish between $f_k$, for random key $k$, and a random function $r$ from $\{0,1\}^n$ to $\{0,1\}^l$

$\mathbb{ADV}$

Guess of $a$

$g_a = f_k$

$g_{\tilde{a}} = r$

$a \in_R \{0,1\}, k \in_R K, r \in_R \{ \text{fun}: \{0,1\}^n \rightarrow \{0,1\}^l \}$
Fixed Input Length PRF and MAC

- Every FIL PRF is also a FIL MAC.
  - Proof sketch: it is not feasible to find a forgery in a random function.

- Practical candidates:
  - Compression functions of MD-constructed hash
    - MD5, SHA-1, RIPE MD,…
  - Block ciphers
    - DES, Triple DES, AES,…
  - Natural design: CBC mode…
Recall: Cipher Block Chaining (CBC)

Split plaintext $m$ to blocks $m[1], \ldots$

Initialization Vector (IV) random number (sent `in clear``)

Key $k$ of $E$ and of $CBC[E]$

Ciphertext $c[0], c[1], c[2], \ldots$

$MAC_k(m)$
CBC MAC [BKR94]

- Let: $\text{CBC}_k(\{m_i\}) = E_k(m_1 \oplus E_k(\ldots E_k(m_1 \oplus IV))))$
  - Widely deployed standard, especially in banking
  - With DES, vulnerable to key guessing

- Claim 2 [BKR94]: if $E$ is a PRP (or 1-block FIL-PRF), then $\text{CBC}_k$ is an $l$-block FIL-PRF (and an $l$-block FIL-MAC) [Proof omitted, see BKR94]

- But we want variable-length input…

![Diagram of CBC MAC](attachment:image.png)

$MAC_k(x[1]..x[l])$
VIL Pseudo-Random Function

- **VIL-PRF:** a collection of efficient functions \( \{f_{k,m} : \{0,1\}^m \rightarrow \{0,1\}^l\} \), such that no adversary can efficiently distinguish between \( f_{k,m} \), for random key \( k \), and a random function \( r \) from \( \{0,1\}^m \) to \( \{0,1\}^l \).

\[
\begin{align*}
a &\in_R \{0,1\}, \text{key} \in_R K, \ r \in_R \{ \text{fun}: \{0,1\}^m \rightarrow \{0,1\}^l \} \\
g_a &= f_{k,m} \\
g_{\tilde{a}} &= r
\end{align*}
\]
CBC MAC is not secure for arbitrary variable-length inputs.

Example: adversary asks to receive
\[ b = CBC_k(a) = E_k(IV \oplus a) \]; then output \(<ac,b>\)
where \(c = a \oplus IV \oplus b\). This is right since
\[ CBC_k(ac) = E_k(c \oplus E_k(a \oplus IV)) = E_k(a \oplus IV) = b. \]

Appending the length as last block \(CBC_k(\|a\|, a)\)
(like in MD construction) does not help.

But... prepending the length \(CBC_k(\|a\|, a)\)
works!
Claim: Variable Input Length MAC

- **Claim 3**: $CBC_k$ is a VIL-MAC if inputs are *prefix-free*
  - Namely: CBC MAC is secure if no input is a prefix of another
  - Proof: see [BKR94]

- Possible implementation: prepend length
  - Given a family of secure $n \times l$ FIL-MAC for every length $n$: $MAC_{k,n}(m)$ (e.g. $CBC_k(m)$)
  - Let $MAC'_k(m) = MAC_{k,n}(|m|, m, pad)$ s.t.
    - $n = \lceil (|m| + \log_2(|m|))/l \rceil$, $pad = \{0\}^{\lfloor |m| + \log(|m|) - nl}$
  - $MAC'$ is a secure VIL-MAC.
Performance of CBC MAC

- MAC of \( n \) blocks requires \( n \) PRF evaluations
- Typically implemented with block cipher
- Improving speed:
  - Parallelize MAC computation
    - Recent results, see e.g. [BR02] (not covered here)
  - Using faster crypto-functions as PRF
    - Hash functions: faster than ciphers
    - Compression-functions of hash: faster yet
Hash based MAC

- Advantages:
  - Higher speed
    - Most designs based on compression functions
  - Hash functions are widely, freely available
  - Specific HMAC construction now standardized

- Heuristic constructions:
  - $MAC_k(m) = h(k || m)$
  - $MAC_k(m) = h(m || k)$
  - $MAC_k(m) = h(k || m || k)$
  - Secure under `random oracle analysis`
Recall: Random Oracle Analysis

- Analyze as if \( h() \) is selected as a *random function*
  - Of course an invalid assumption as \( h() \) is fixed!

- For example: \( MAC_k(m) = h(k \| m) \)
  - Assume \( h \) is a random function
  - Chosen plaintext \( m_i \) sets \( h(k \| m_i) \)
  - The value of \( h(k \| m) \) (not one of \( m_i \)) remains random

- This precludes generic attack on heuristic hash-based MAC

- But could be an attack with *specific* \( h \)...
Analysis for hash based MAC

- **Goal:** weaker assumptions about $h$
- **Basis for HMAC construction** [BCK]
  - Widely deployed MAC standard
- **Provable secure and practical construction**
  - MAC forbids any forgery, not `just` collision
  - Hash has no secret key $\rightarrow$ `easier` to attack
  - Define *collision-resistant-only MAC (CR-Only MAC)*: secret key, adversary wins only if it finds a collision
    - Called Weak *keyed* CRHF in [BCK]
  - Weaker assumption than either MAC or Hash!
    - Every secure MAC, CRHF `is` a CR-Only MAC
Collision-resistant-only MAC
(In [BCK]: Weak keyed CRHF)

- A **Collision-resistant-only MAC**: adversary cannot efficiently find collision, given \( h_k(m_i) \) for \( \{m_i\} \) chosen by the adversary (for random key \( k \)).

- Weak collision requirements:
  - Key \( k \) is secret, adversary can’t compute hash function.
  - Finding collisions by computing many hashes becomes infeasible.
Every MAC is a Collision-resistant-only MAC
- Since every collision is also a forgery

But Collision-resistant-only MAC is not always MAC
- A forgery \( m, h_k(m) \) may be found without finding any collision \((x, x' \text{ s.t. } h_k(x) = h_k(x'))\).

Can we construct VIL MAC from VIL CR-only MAC and FIL MAC?

Motivation: implement...
- FIL MAC by `keying` compression function (FIL hash function – used in MD construction of VIL hash),
- VIL Collision-resistant-only MAC by `keying` a hash function

`Keying` - e.g. by using the key as IV
Recall: Merkle-Damgard Construction

- Build \( h \) from compression function: \( c : \{0,1\}^{2n} \rightarrow \{0,1\}^n \)
  - Compression function is a FIL hash function
- Let the input \( x \) be \( b \) blocks of \( n \) bits
  - Pad the last block if necessary
  - Add extra block, \( x[b+1] = |x| \)
- Let \( y_0 = IV \) be some fixed \( n \) bits (IV=Initialization Value)
- For \( i = 1, \ldots, b+1 \), let \( y_i = c(x[i], y_{i-1}) \)
- Output \( h(x) = y_{b+1} \)
**Nested MAC Construction**

- **Use keyed hash function:**
  - Use the IV as key (usually simple to change the IV)
  - Let $h_k$ denote the MD construction with $IV=k$ and compression function $c$

- **Nested MAC (NMAC) Theorem [BCK96]:** If:
  - Compression function $c$ is a FIL-MAC (fixed input length)
  - $h_k$ is a Collision-resistant-only VIL MAC

Then $NMAC[k_1,k_2](x)=h[k_1](h[k_2](x))$ is secure MAC.

![Diagram](image)

\[ NMAC[k_1,k_2](x) = h[k_1](h[k_2](x)) \]
Nested MAC Construction

**Nested MAC (NMAC) Theorem** If:
- Compression function \( c \) is a secure FIL-MAC
- \( h_k \) is a Collision-resistant-only MAC

Then \( \text{NMAC}[k_1,k_2](x) = h[k_1](h[k_2](x)) \) is secure MAC.

- \(|h()|=1\) block \(\rightarrow c[k_1](h[k_2](x)) \approx h[k_1](h[k_2](x))\)
- Requires compression to hide the key, while IV (for hash) was not hidden
Theorem [BCK96]:
If $c$ is FIL-MAC and $h_k$ is Collision-resistant-only MAC, then $NMAC_{k_1,k_2}(x)$ is MAC.

Proof: Let $A_N$ be an efficient adversary which finds forgery in $NMAC$.

Forgery: $(x,y)$ s.t. $x \neq x_i$ and $y = NMAC_{k_1,k_2}(x)$
Nested MAC Construction — Proof

- **Theorem’ [BCK96]:** If $c$ is FI-MAC and $h_k$ is Collision-resistant-only MAC, then $\text{NMAC}’_{k1,k2}(x) = c_{k1}(h_{k2}(x))$ is MAC.

- **Proof:** Let $A_N$ be an efficient adversary which finds forgery in $\text{NMAC}’$. 

\[ \text{Forgery: } (x,y) \text{ s.t. } x \neq x_i \text{ and } y = c_{k1}(h_{k2}(x)) \]
Proof (cont’): We define alg. \( A_{ch} \), using \( A_N \) and \( c_{kl} \) as oracles (‘black box’); \( A_{ch} \) finds collision in \( h_{k2} \) or forgery in \( c_{kl} \), for unknown \( k_l \).

\[
(x, y = c_{kl}(h_{k2}(x)))
\]

collision in \( h_{k2} \) or forgery in \( c_{kl} \)
NMAC – Proof (cont)'

$(x,y)$ is a forgery i.e. $y=c_{k1}(h_{k2}(x))$ and $x \neq x_i$. If $h_{k2}(x)=h_{k2}(x_i)$, for some $i$, this gives collision in $h_{k2}$. Let $r=h_{k2}(x)$. Assuming no collision, $r \neq r_i$; but $y=c_{k1}(r)$, i.e. $A_{ch}$ found forgery in $c_{k1}$.

$k_2 \in_R \{0,1\}^n$

For $i=1,\ldots$ do:

- $x_i \leftarrow A_N$
- $r_i \leftarrow h_{k2}(x_i)$
- $A_N \leftarrow c_{k1}(r_i)$

$(x,y) \leftarrow A_N$

Output $(h_{k2}(x),y)$

Collision in $h_{k2}$ or forgery in $c_{k1}$
Implementing NMAC requires hash function with variable IV

HMAC is a variant allowing use of keyless (fixed IV) hash functions `as is`:

\[ \text{HMAC}_k(x) = h(k \oplus \text{opad} \parallel h(k \oplus \text{ipad} \parallel x)) \]

- opad, ipad are constants selected to maximize the hamming distance between \( k \oplus \text{opad} \) and \( k \oplus \text{ipad} \). Specifically, opad is a string of x'36' bytes, and ipad is a string of x'5c' bytes.

Widely deployed standard [RFC2104]
HMAC - Hash based MAC

- HMAC uses keyless (fixed IV) hash functions:
  \[
  HMAC_k(x) = h(k \oplus opad \| h(k \oplus ipad \| x))
  \]

- Similar to NMAC but fixed IV – key is in input:
  - Let \( k_1 = c_{IV}(k \oplus opad) \), \( k_2 = c_{IV}(k \oplus ipad) \)
  - \( HMAC_k(x) = NMAC_{k_1,k_2}(x) \)

- Security argument: \( k_j = c_{IV}(k \oplus opad) \) is ‘almost random’; security follows from NMAC.
Using MAC:

Shared Key Mutual Authentication

- Model: Alice and Bob share secret *master key* $k$
- Goals
  - Mutual authentication: Alice knows it talked with Bob and vice verse.
  - Parties may also send a message; prevent replays.
  - Allow multiple concurrent connections.
  - Either party can initiate.
- Basic problem, appears (and is) easy
- …but also easy to do wrong

Skip mistakes

- SNA – IBM’s Secure Network Architecture
  - Predominant network protocol till late eighties
- Protocol: \((N_a, N_b \text{ - randomly chosen nonces})\)

![Diagram of the SNA LU6.2 Protocol]

\[N_a \rightarrow N_b, E_k(N_a)\]

\[E_k(N_b) \leftarrow\]
**Attack on SNA LU6.2 Authentication**

- **Idea:** Eve opens **two** connections with Bob... sending $N_b$ to Bob in 2\textsuperscript{nd} connection to get $E_k(N_b)$
Conclusions & Thumb-rules

- Prevent re-direction of message to sender
  - Identify party in challenge

- Prevent re-direction of flow $i$ to flow $j \neq i$
  - Ensure different flows are easily distinguished

- Prevent use of old challenge
  - Select new random challenge (nonce) or time

- Do not compute values chosen by Adversary
  - Include self-chosen nonce in the protected reply

- Authenticate with MAC, not encryption
Two Party Protocol (2PP) [BGH*93]

- Fixed SNA protocol
- Use MAC rather than encryption to authenticate
- Separate 2nd and 3rd flows – 3 vs. 2 input blocks
- Include identities \((A,B)\) to prevent redirections
- Proof of security: from MAC properties (Claim 1)
  - See [BR93] for definition and proof

```
Na, MACk(Na, Nb, A||B) = 1
```

```
Na, MACk(Na, Nb)
```

```
Nb, MACk(Na, Nb, A||B)
```

```
MACk(Na, Nb)
```

\(A\) and \(B\) are identities.
Authenticating messages

- Optionally, authenticate messages $m_A, m_B$ by including their hash in the MAC inputs
- To authenticate many messages (in order):
  - Add sequence numbers
  - Can use same nonces for multiple messages

Alice \[ N_a \]

Bob

$N_b, \text{MAC}_k(N_a, N_b, h(m_B), A||B)$

MAC$_k(N_a, N_b, h(m_A))$
Efficient Implementation with CBC MAC

- Assume: one block per parameter
  - $MAC_k(N_a, N_b) = E_k(N_b + E_k(N_a))$
  - $MAC_k(N_a, N_b, B) = E_k(A\|B + E_k(N_b + E_k(N_a)))$
- Potential reuse: $MAC_k(N_a, N_b, B) = E_k(B + MAC_k(N_a, N_b))$
  - Only three `block operations` for entire protocol
- Suggested in [BGH*93]
Implementation with CBC MAC

- Is this secure?
  - Claim 3 (foil 26) [BKR94] shows CBC is a MAC if inputs are prefix-free
  - But here 3rd flow is prefix of 2nd flow – not prefix free!
  - Seems secure… but I’m not aware of proof

\[ N_a \]

\[ N_b, E_k(A\|B+E_k(N_b+E_k(N_a))) \]

\[ E_k(N_b+E_k(N_a)) \]
Question: can 2PP authenticate users?

- Is 2PP secure using a password for the key $k$?
- Problems:
  - Password is not uniformly distributed
  - Limited number of common passwords – attacker can guess (Dictionary attack)

![Diagram of 2PP authentication process]

- $N_a$
- $N_b, MAC_k(N_a, N_b, A \| B)$
- $MAC_k(N_a, N_b)$
Using 2PP with passwords

- Problem: Password is not uniformly distributed
- Solution: use $pw' = h(pw)$ where $h$ is a resilient PR hash
  - Ok as long as password contains `enough random bits`
- Problem: Dictionary attack
  - Limited number of common passwords – attacker can guess
  - Attack uses only one recorded login (passive!)
- Solutions: Encrypt entire exchange…
  - Using key shared between terminal and host
  - Using host’s public key (randomized encryption!) – next lecture
    - Usually best solution; used e.g. by SSL/TLS
  - By first establishing new random key, then authenticating it…
    - Advanced login protocols – not in this course
Authentication and Encryption

- Usually to secure communication we need:
  - Encryption – for confidentiality
  - Message authentication – for integrity
- How to achieve both goals?
  - Cryptosystems/ciphers only encrypt
  - MAC only authenticate
- Standard solution: use both MAC and encrypt
  - Some works on combined Authentication (MAC) and Encryption for efficiency
Authentication and Encryption

- Use both Encrypt and MAC to protect confidentiality+integrity

Questions:
- Which keys to use for Encryption, MAC?
  - Can we use same keys for both?
  - Must we use two independent keys (double key length)?
- How to combine MAC, encryption?
Can we use same key for MAC + Encryption?

- No.. This may break either/both functions
- Example: consider using:
  - CBC Authenticate then Encrypt (AtE)
  - Authenticate: CBC MAC
    - \( a = \text{CBC} \quad \text{MAC}_k(m_1m_2m_3) = E_k(m_3 \oplus E_k(m_2 \oplus E_k(m_1))) \)
  - Then, CBC Encryption:
    - \( C_k(m_1m_2m_3) = \text{CBC}_k(m_1m_2m_3 || a) \)
Using same $k$ for CBC auth, encrypt...

Split plaintext $m$ to blocks $m[1],...$

Initialization Vector (IV) random number (sent `in clear`)

Key $k$ of $E$ and of $CBC[E]$

Ciphertext $c[1],c[2]...$

- $c[i]=E_k(m[i] \oplus c[i-1])$
- $a=CBC-MAC_k(m_1m_2m_3)=E_k(m_3 \oplus E_k(m_2 \oplus E_k(m_1)))=c[3]$
- $c[4]=E_k(a \oplus c[3]))=E_k(0)$
Keys for MAC + Encryption

- Conclusion: better not use same key for authentication and MAC.
- Must the two keys be independent?
  - Some overhead of double key length
  - Kerberos V5: $k_{mac} := k_e \oplus \text{F0F0...F0}$
    - Exercise: demonstrate $E, MAC$ where this fails
  - Better: use pseudo-random keys
  - Let $k$ be the shared key; use:
    - $k_{e,a \rightarrow b} = \text{PRP}_k(“Encrypt,a \rightarrow b”)$
    - $k_{a,a \rightarrow b} = \text{PRP}_k(“Authenticate,a \rightarrow b”)$
    - Similarly, keys for traffic from $b$ to $a$ ($b \rightarrow a$)
Use both Encrypt and MAC to protect confidentiality+integrity

How to combine?

SSH authenticates and encrypts (A&E):
- \( C=Enc(m), A=MAC(m) \), send \((C,A)\)
- Not secure… Why? MAC may expose (some of) \( m \).

SSL authenticates, then encrypts (AtE):
- \( A=MAC(m), C=Enc(m,A) \), send \( C \)

IPSEC encrypts, then authenticates (EtA):
- \( C=Enc(m), A=MAC(C) \), send \((C,A)\)

Which is better?
SSL authenticates, then encrypts (AtE):
- \[ A = \text{MAC}(m), \ C = \text{Enc}(m,A), \text{ send } C \]

IPSEC encrypts, then authenticates (EtA):
- \[ C = \text{Enc}(m), \ A = \text{MAC}(C), \text{ send } (C,A) \]

Encrypt then Authenticate (EtA) is better:
- Prevention of chosen ciphertext attacks (on cipher)
- Reject unauthenticated messages w/o decryption
  - To foil `clogging` attacks
- Proof of security of combined mechanism
  - Assumption: attacker knows if authentication failed or not.
  - [CK01]: encrypt-then-authenticate (EtA) is secure
  - [K01]: counterexample to AtE
  - AtE OK for most encryptions, e.g. CBC, One time pad
Attack on Authenticate-then-Encrypt

- Define the following cipher $E$ based on One-Time Pad (OTP) (or a pseudo-random generator):
  - $E_k(x) = \text{Transform}(x) \oplus k$ \textit{[bit-wise XOR]}
  - \textit{Transform} each bit of the plaintext to two bits:
    - Zero bits (0) are transformed to two zeros (00)
    - One bits (1) are transformed to (01) or (10) randomly
  - $E$ indistinguishable under chosen plaintext attack
  - We show an attack on \textit{auth-then-encrypt} when using $E$
  - \textit{Attack}: flip first two bits of ciphertext.
    - If authentication is still valid, first plaintext bit is 1
    - If authentication fails, first plaintext bit is zero.
Conclusion

- **MAC – Message Authentication Code**
  - Sender appends `authenticating tag` (MAC) to message
  - Recipient verifies tag using shared secret key
  - HMAC – standard MAC, highly efficient, based on hashing.

- **Use MAC to authenticate communication**
  - Add identities, flow #, your challenge to messages

- **Authentication + Encryption protects channel**
  - Encrypt then authenticate (EtA) is secure