Symmetric and Password-based encryption

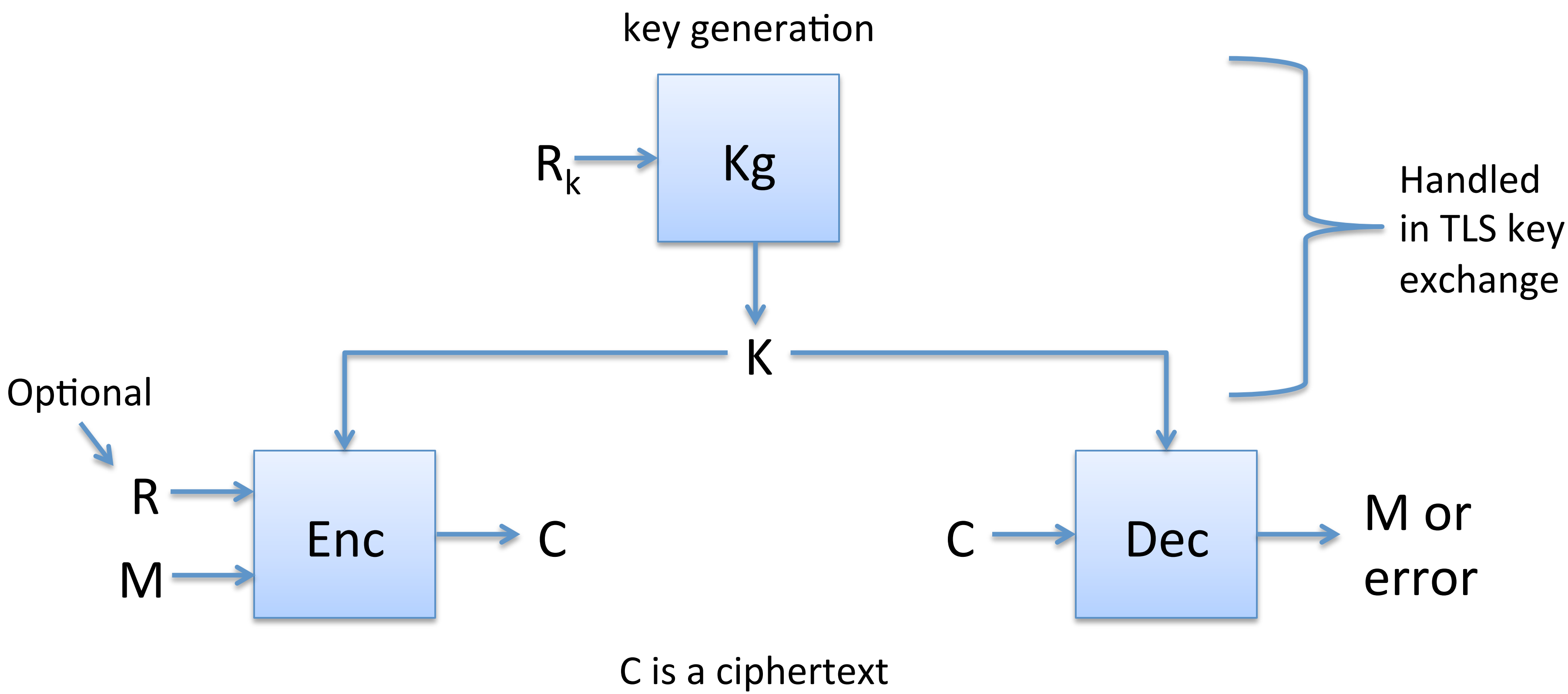
CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

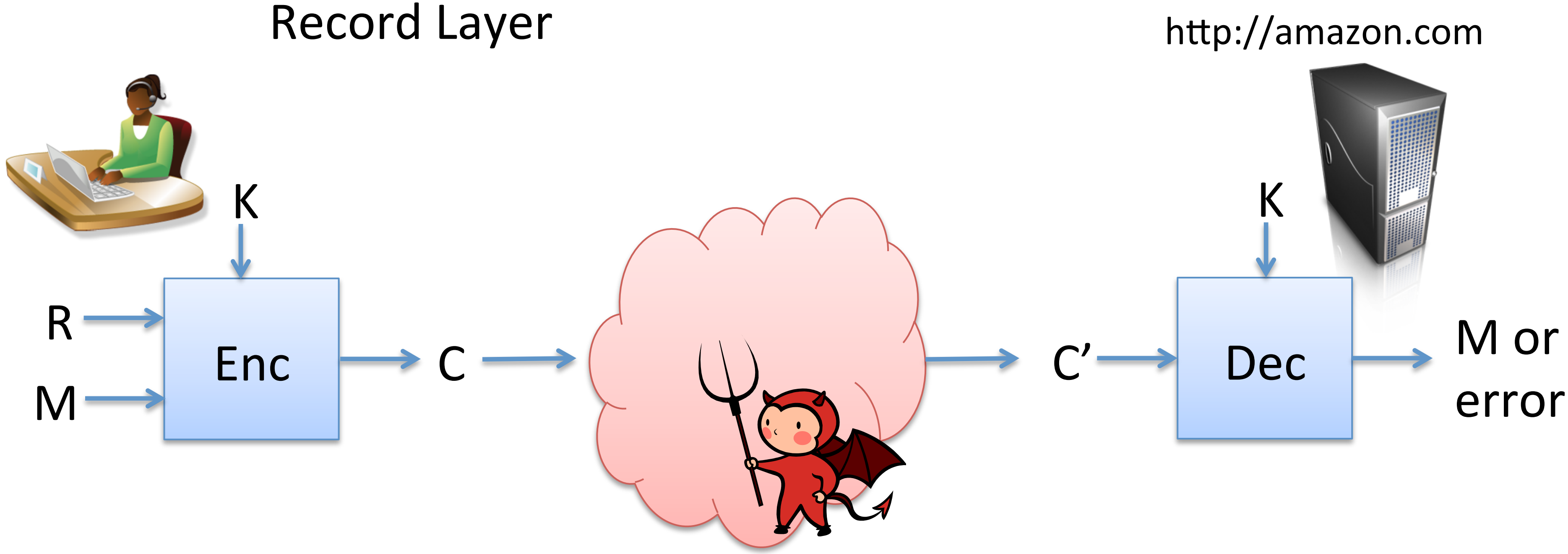
rist at cs dot wisc dot edu

Symmetric encryption



Correctness: D(K , E(K,M,R)) = M with probability 1 over randomness used

In TLS symmetric encryption underlies the Record Layer

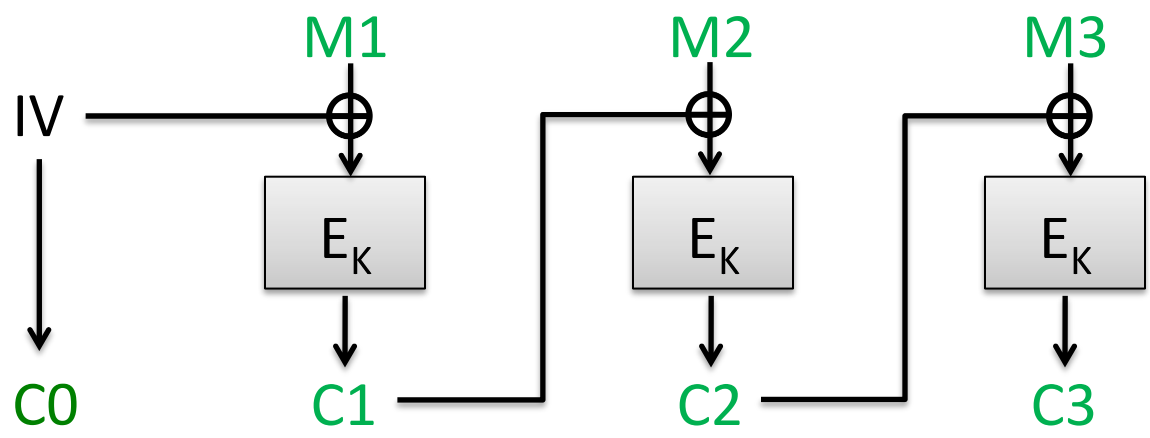


What security properties do we need from symmetric encryption?

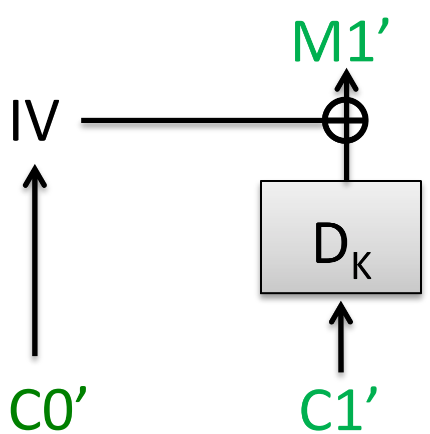
1) Confidentiality: should not learn any information about M
2) Authenticity: should not be able to forge messages

Often referred to as Authenticated Encryption security

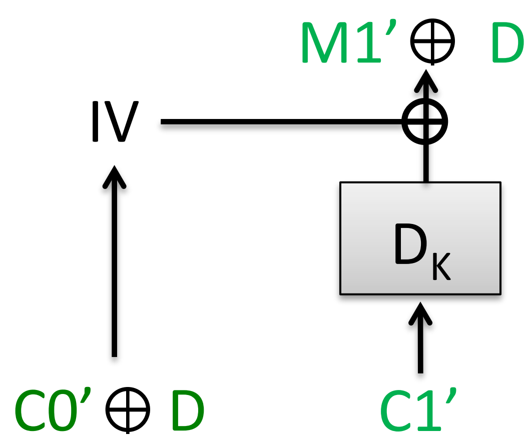
Active security of CBC mode



What about forging a message? Pick any C0’, C1’ …

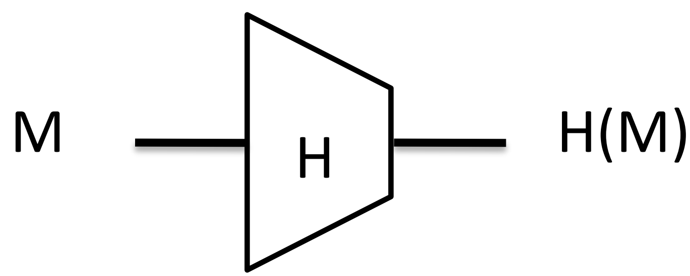


Better yet
for any D:



Hash functions and message authentication

Hash function H maps arbitrary bit string to fixed length string of size m



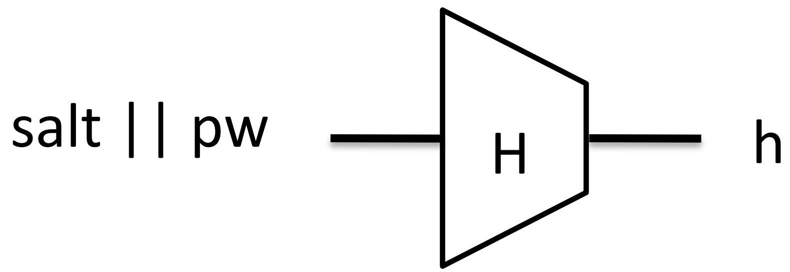
MD5: $m = 128$ bits
SHA-1: $m = 160$ bits
SHA-256: $m = 256$ bits

Some security goals:

- collision resistance: can't find M != M' such that H(M) = H(M')
- preimage resistance: given H(M), can't find M
- second-preimage resistance: given H(M), can't find M' s.t. H(M') = H(M)

Hash function application example

Password hashing. Choose random salt and store (salt,h) where:



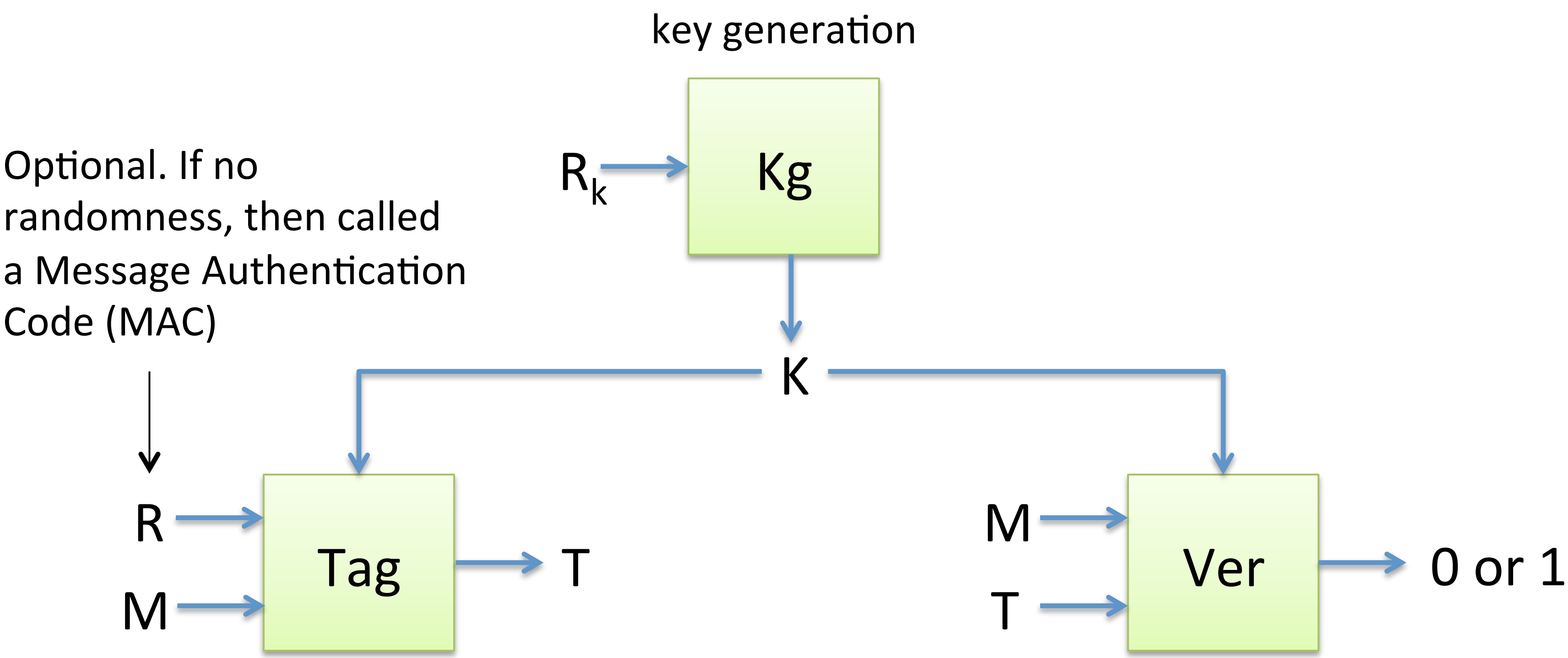
The idea: Attacker, given (salt,h), should not be able to recover pw

Or can they?

For each guess pw':
    If H(salt||pw') = h then
        Ret pw'

Rainbow tables speed this up in practice by way of precompution. Large salts make rainbow tables impractical

Message authentication



Correctness: Ver(K , Tag(K,M,R)) = 1 with probability 1 over randomness used

Unforgeability: Attacker can't find M',T such that V(K,M',T) = 1

Recall PRF security

$F: \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^n$

Security goal: F(K,M) is indistinguishable from random n-bit string for anyone without K

For $M_1, M_2, \ldots, M_q$ chosen by adversary and distinct

U_i is fresh n-bit uniform string

$F(K,M_1), F(K,M_2), \ldots, F(K,M_q)$

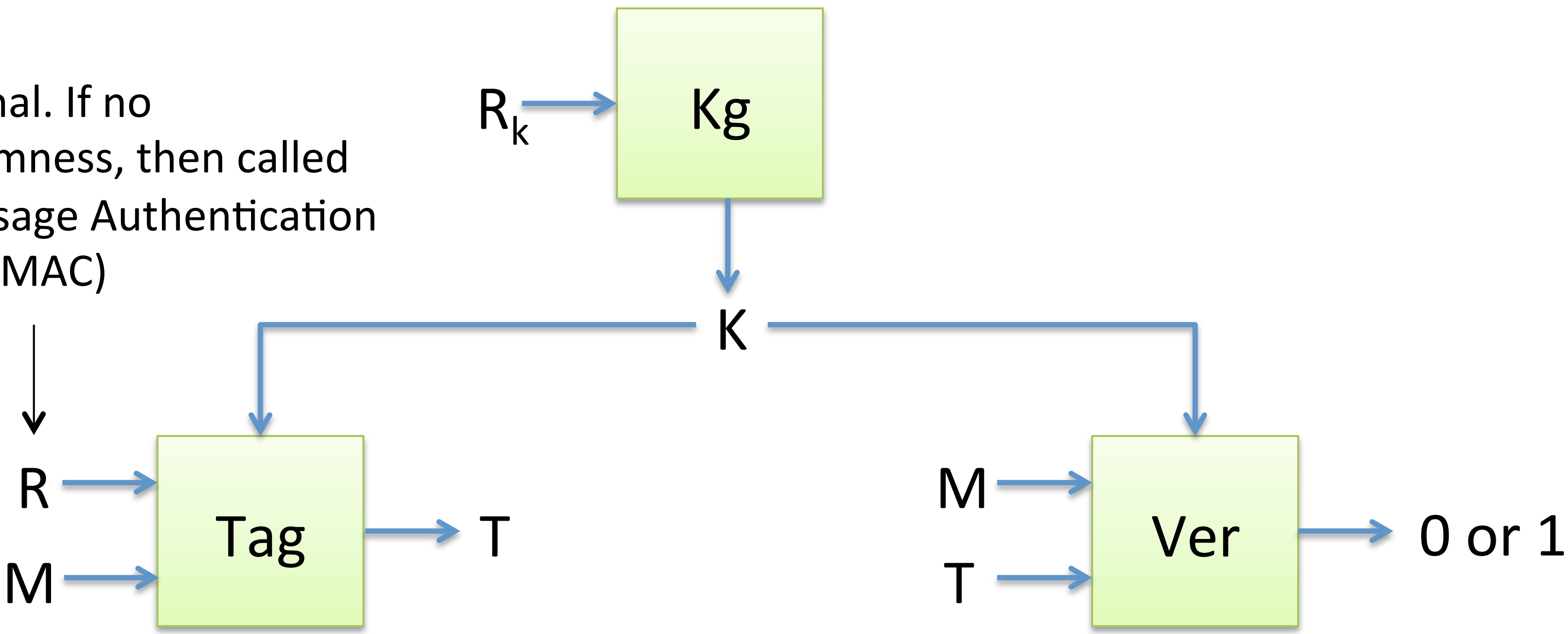
$U_1, U_2, \ldots, U_q$

Adversary that adaptively chooses messages but is limited to "reasonable" q (e.g., $q = 2^{40}$) can't distinguish between two vectors

This means outputs of F are ***unpredictable:***
Given $F(K,M_1), F(K,M_2), \ldots, F(K,M_{q-1})$ no attacker can predict $F(K,M_q)$ with probability $1 / 2^n$ + negligible

Any PRF is a good MAC

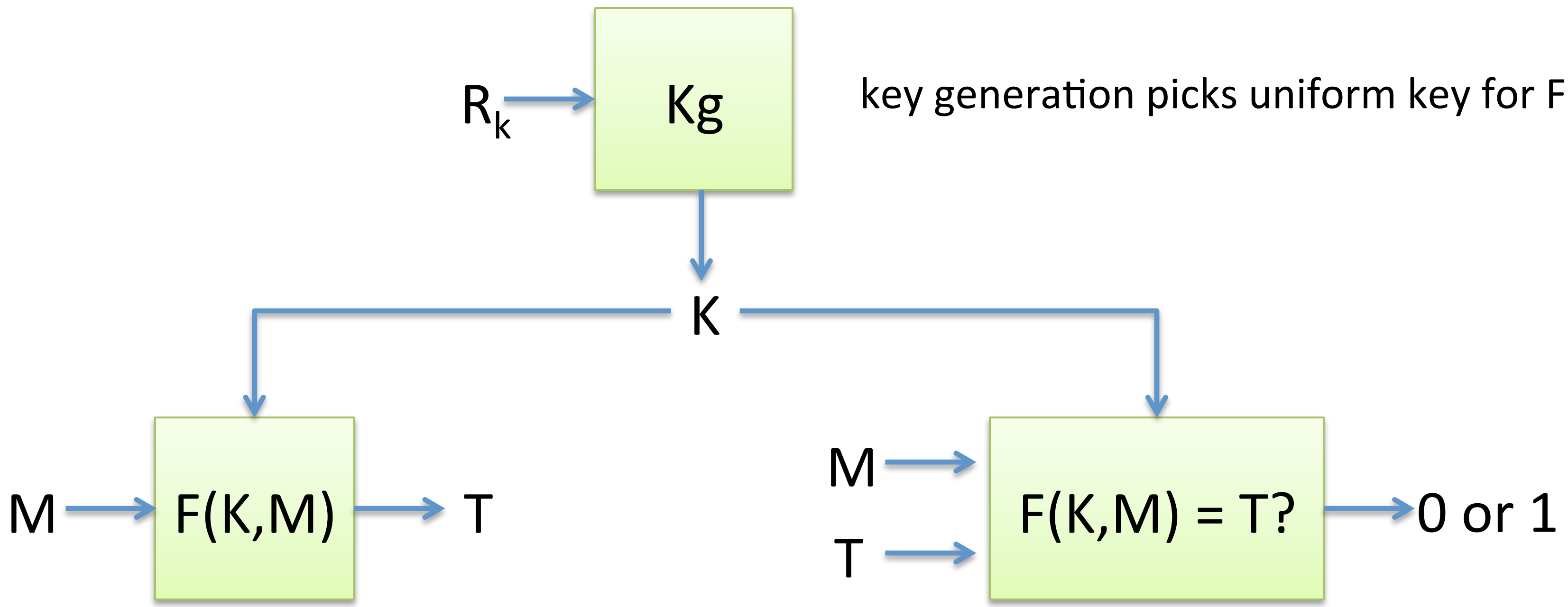
Optional. If no randomness, then called a Message Authentication Code (MAC)



Correctness: Ver(K , Tag(K,M,R)) = 1 with probability 1 over randomness used

Unforgeability: Attacker can't find M',T such that V(K,M',T) = 1

Any PRF is a good MAC



key generation picks uniform key for F

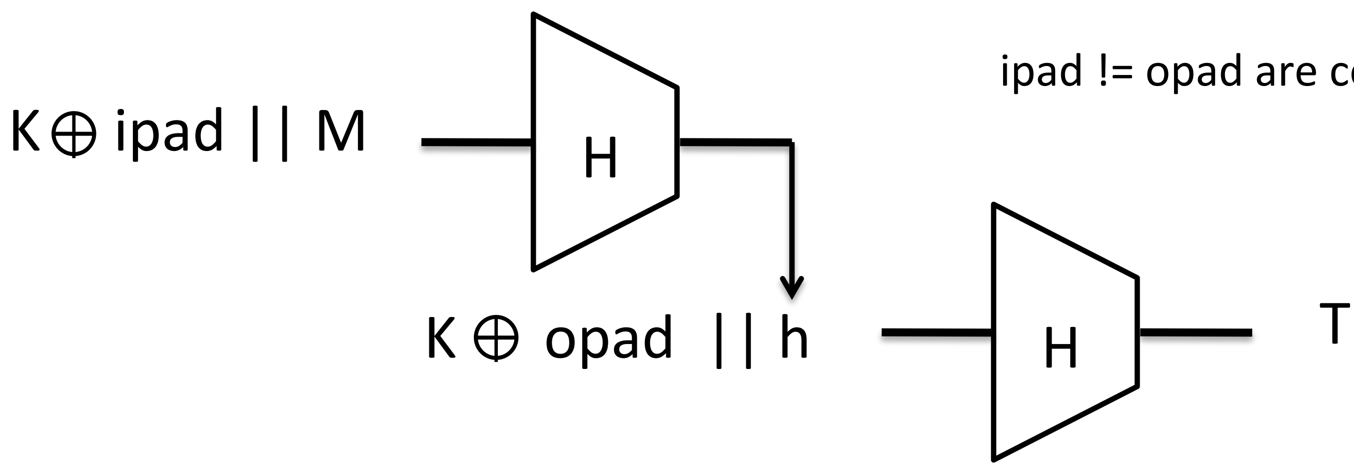
How do we instantiate F?

Message authentication with HMAC

Use a hash function H to build a MAC.
Kg outputs uniform bit string K

Tag(K,M) = HMAC(K,M) defined by:

ipad != opad are constants

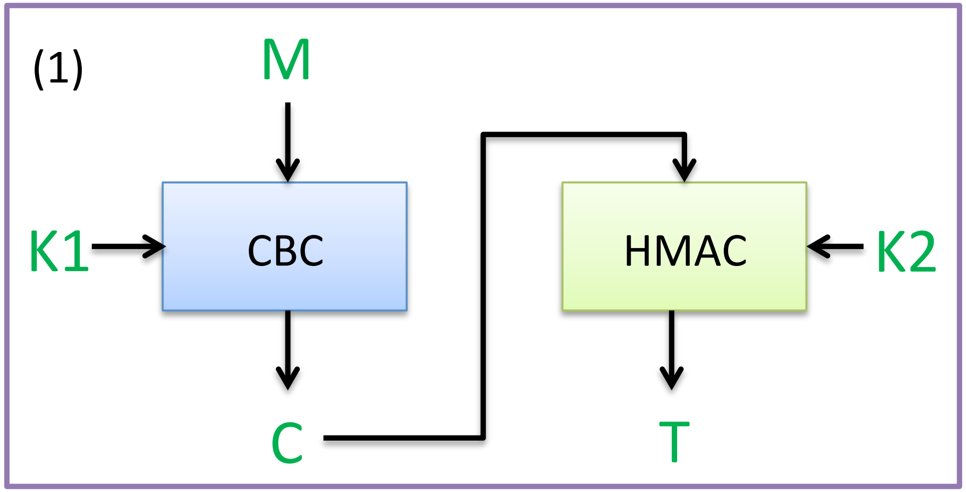


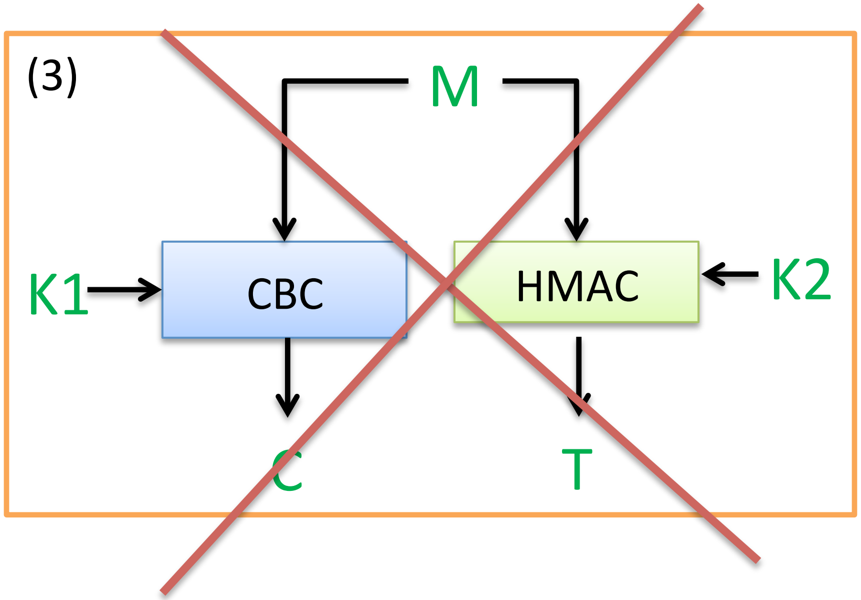
To verify a M,T pair, check if HMAC(K,M) = T

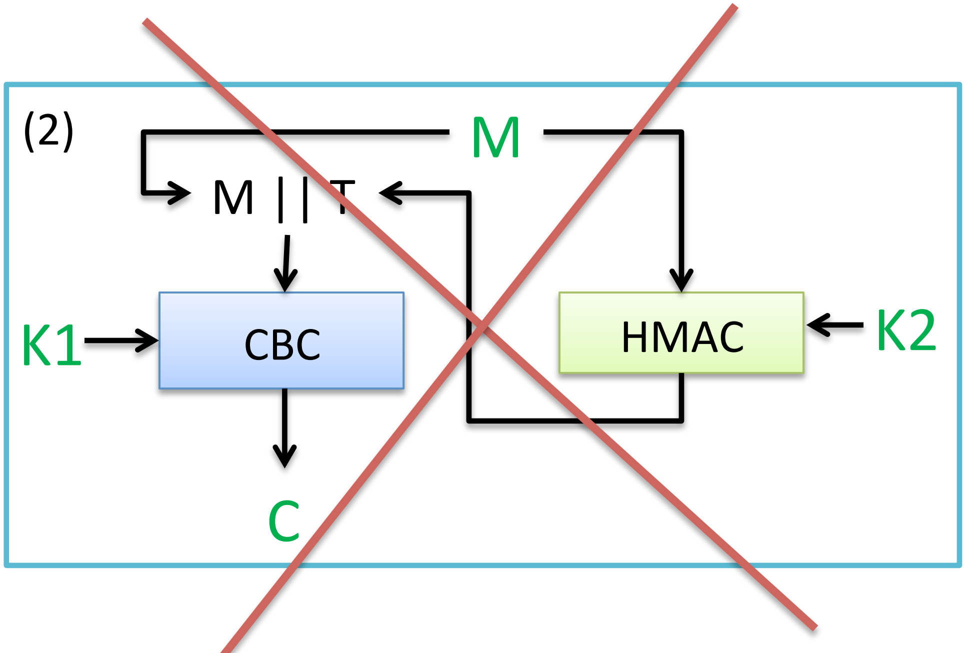
Unforgeability holds if H is a secure PRF when so-keyed

Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

Several ways to combine:
(1) encrypt-then-mac
(2) mac-then-encrypt
(3) encrypt-and-mac

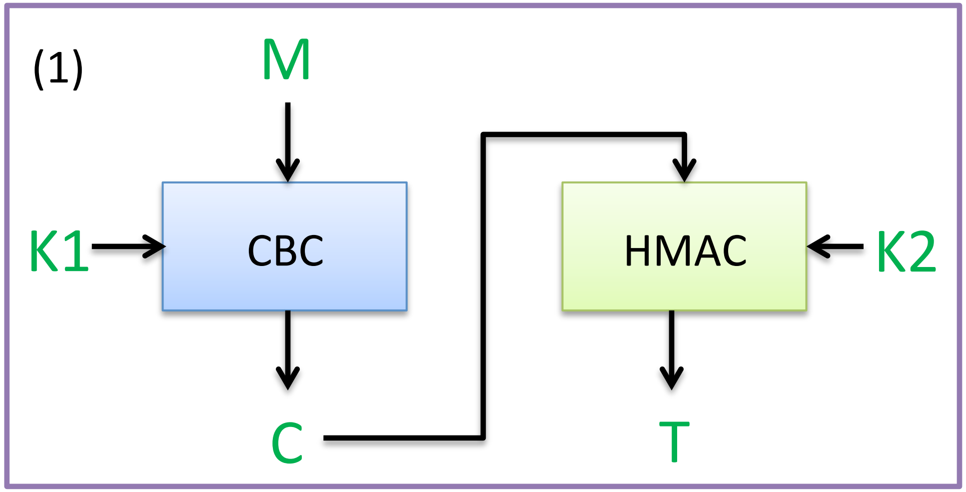






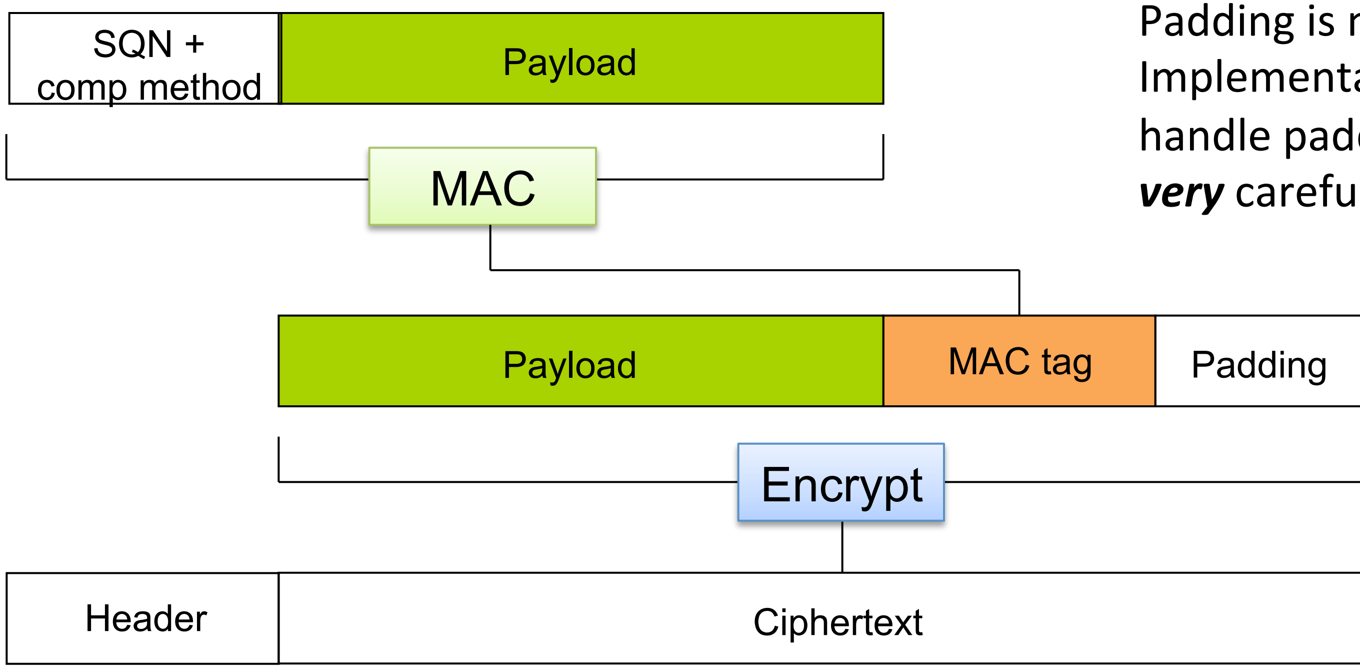
Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

Several ways to combine:
(1) encrypt-then-mac
(2) mac-then-encrypt
(3) encrypt-and-mac



Thm. If encryption scheme provides confidentiality against passive attackers and MAC provides unforgeability, then Encrypt-then-MAC provides secure authenticated encryption

TLS record protocol: MAC-Encode-Encrypt (MEE)



Padding is not MAC'd. Implementations must handle padding checks ***very*** carefully.

MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128

Dedicated authenticated encryption schemes

Attack	Inventors	Notes
OCB (Offset Codebook)	Rogaway	One-pass
GCM (Galios Counter Mode)	McGrew, Viega	CTR mode plus specialized MAC
CWC	Kohno, Viega, Whiting	CTR mode plus Carter-Wegman MAC
CCM	Housley, Ferguson, Whiting	CTR mode plus CBC-MAC
EAX	Wagner, Bellare, Rogaway	CTR mode plus OMAC

Symmetric Encryption Advice

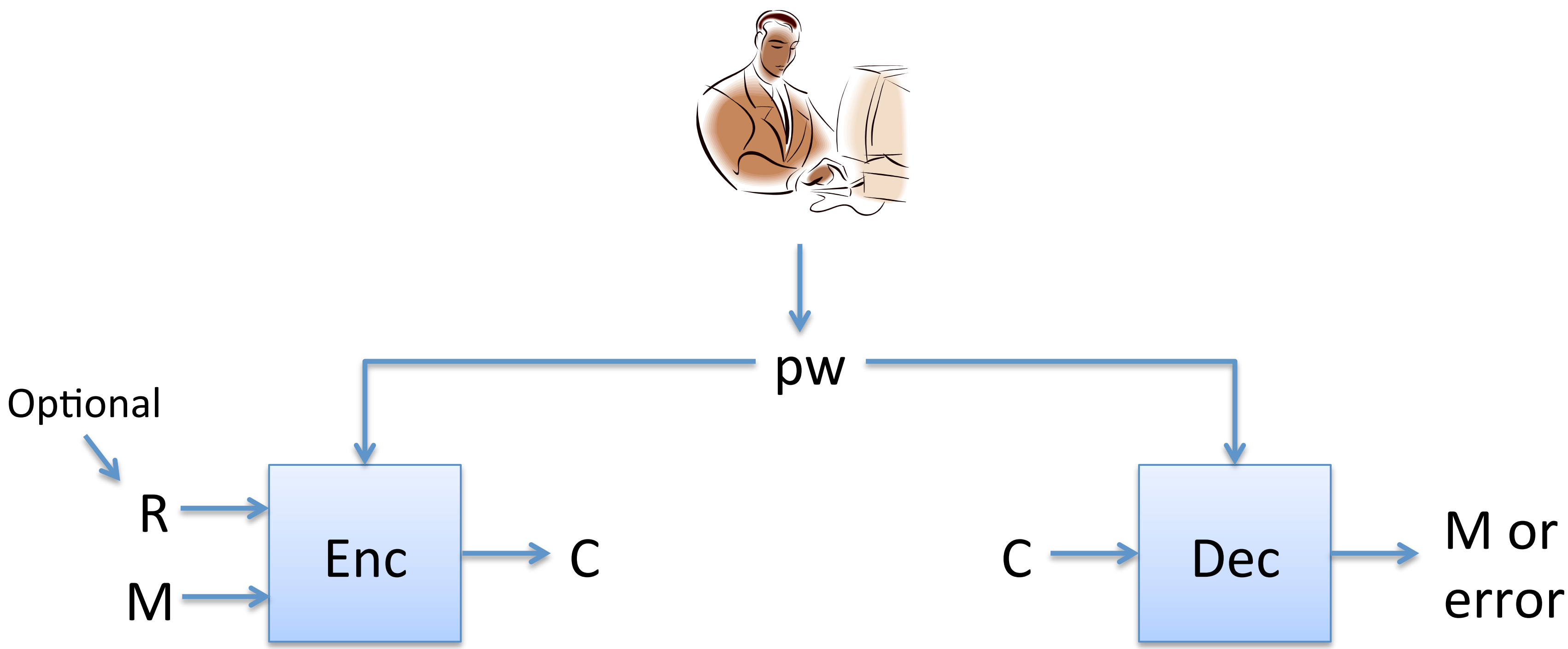
Never use CTR mode or CBC mode by themselves

Passive security is almost never good enough!!

Encrypt-then-MAC better than MAC-then-Encrypt, Encrypt and MAC

Dedicated modes that have been analyzed thoroughly are also good

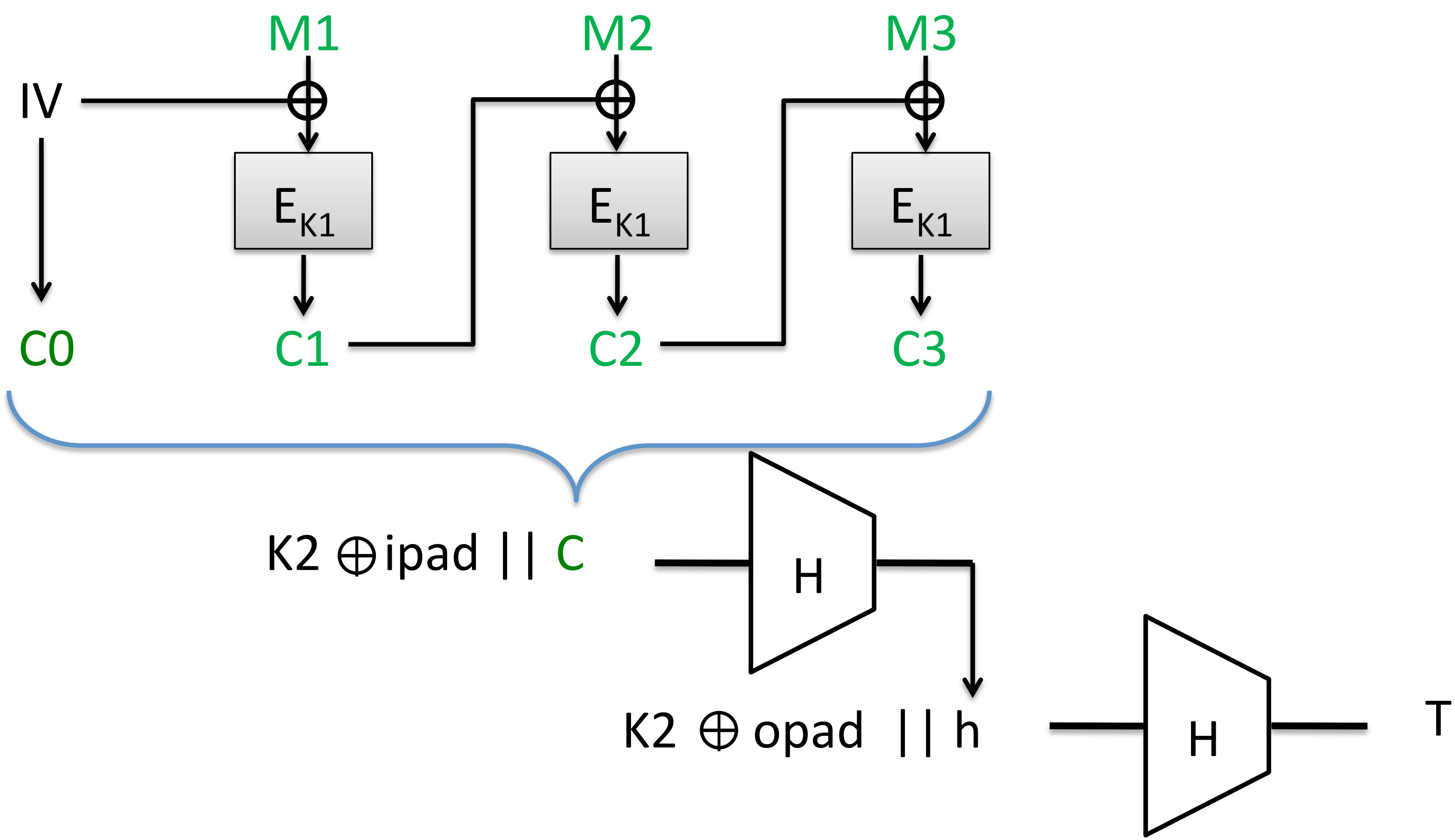
Password-based symmetric encryption



C is a ciphertext

Correctness: D(pw , E(pw,M,R)) = M with probability 1 over randomness used

Encrypt-then-MAC with CBC and HMAC

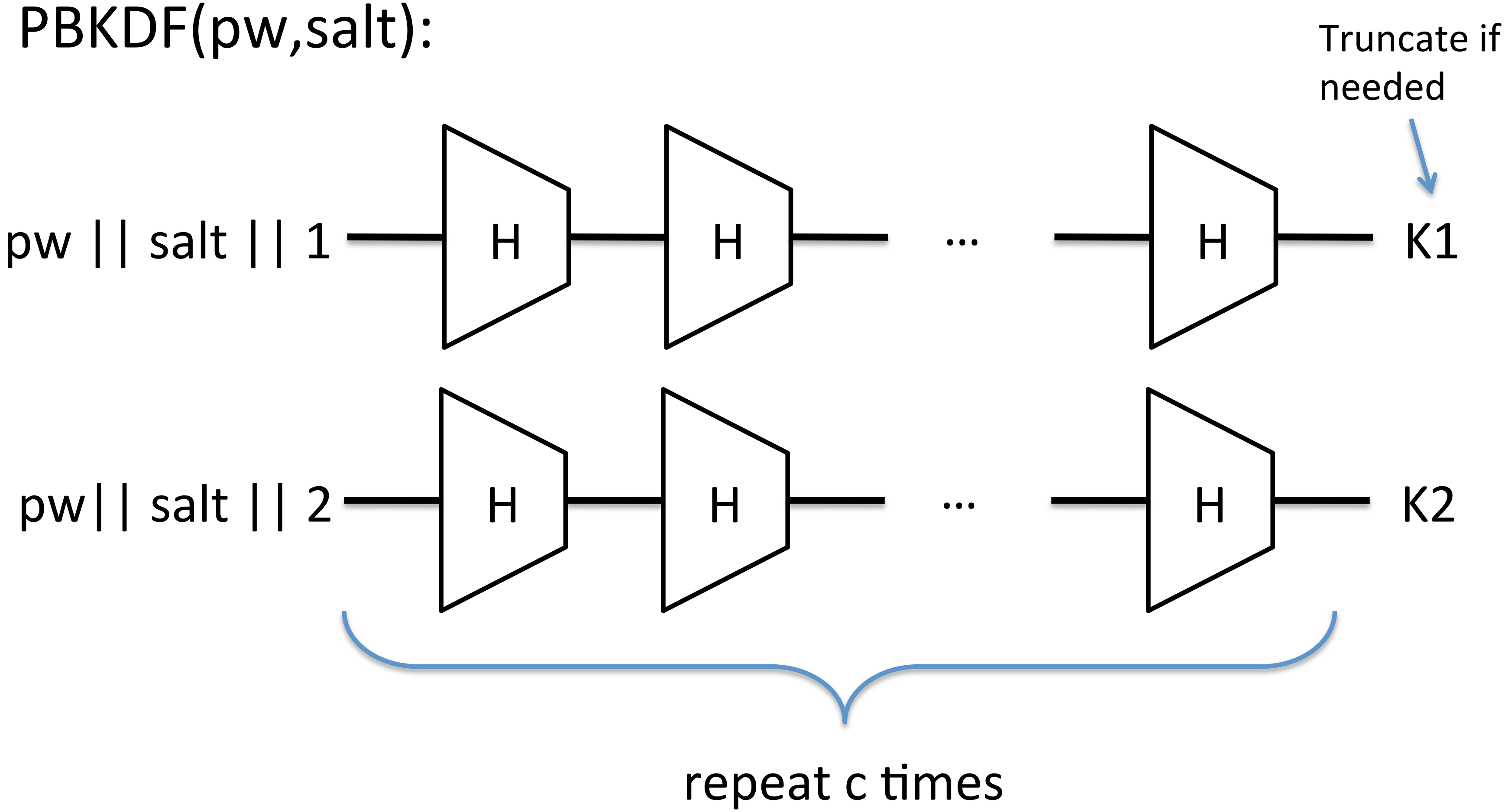


Ciphertext is C,T

How do we use with a pw?

Password-based Key Deriviation (PBKDF)

PBKDF(pw,salt):



PBKDF + Symmetric encryption yields PW-based encryption

<u>Enc(pw,M,R):</u>
salt || R' = R
K = PBKDF(pw,salt)
C = Enc'(K,M,R')
Return (salt,C)

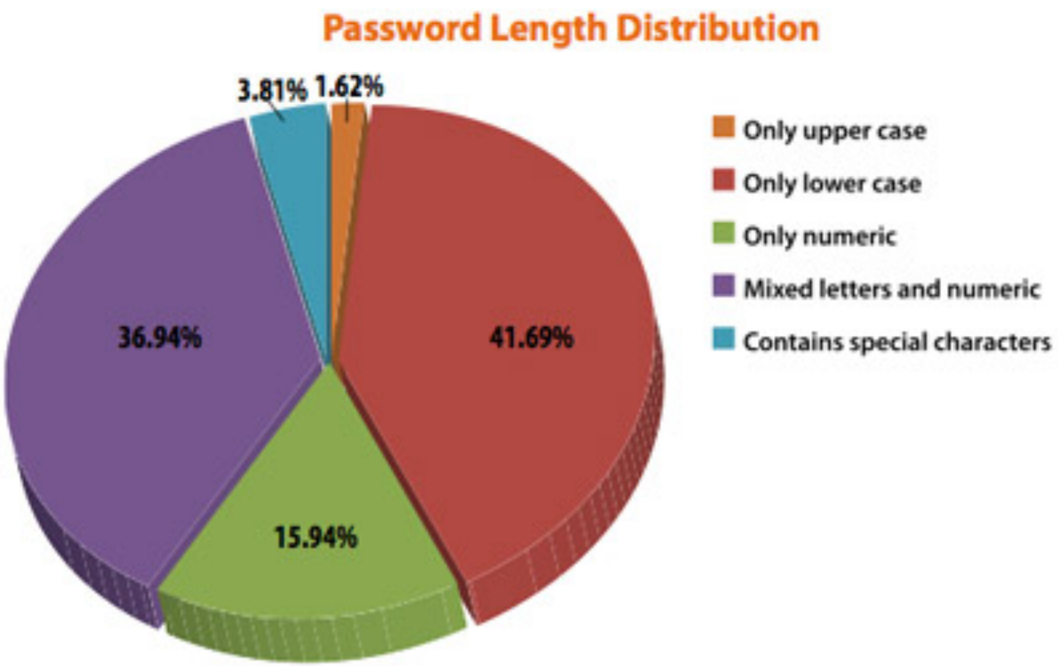
<u>Dec(pw,C):</u>
salt || C' = C
K = PBKDF(pw,salt)
M = Enc'(K,C')
Return M

Here Enc' is a normal symmetric encryption scheme (CBC+HMAC)

Attacks?

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Rank	Password	Number of Users with Password (absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856



From an Imperva study of released RockMe.com password database 2010

Brute-force attacks

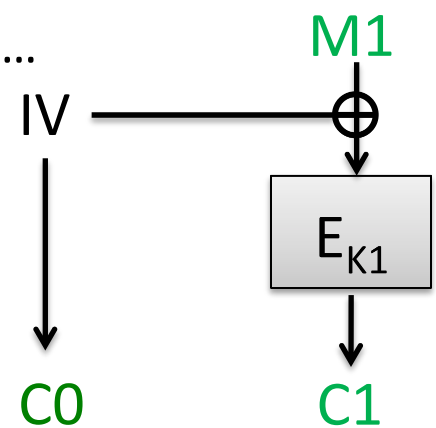
- Given known plaintext, ciphertext pair:
  – M and C = Enc(pw,M)
- Enumerate a dictionary D of possible passwords, in order of likelihood

<u>BruteForce1(M,C):</u>
R || C' = C
foreach pw* in D do
    C* = Enc(pw*,M,R)
    If C* = C' then
        Return pw*

R is salt||IV in CBC-based modes

Both are public:

C = salt || IV || C1 ||…



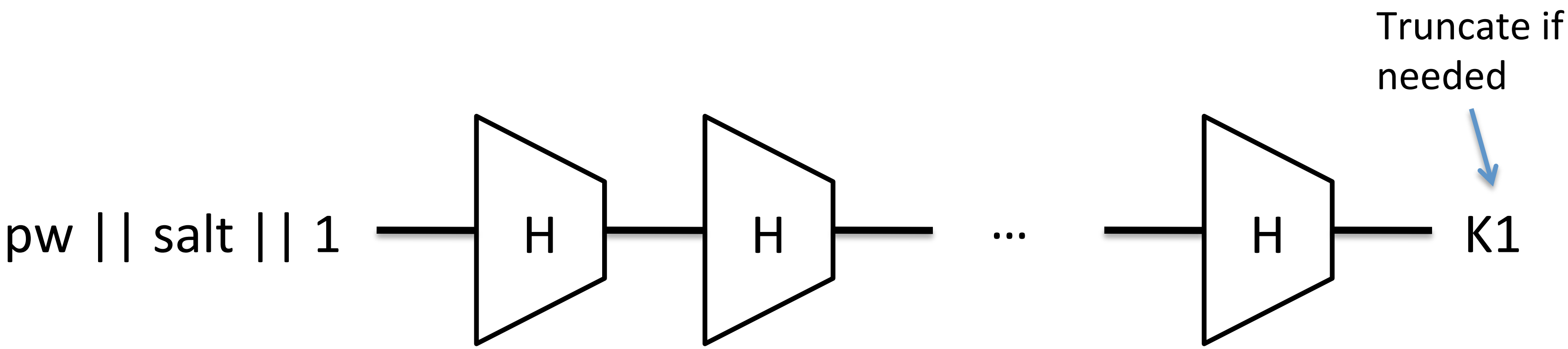
Brute-force attacks

- Given known plaintext, ciphertext pair:
  – M and C = Enc(pw,M)
- Enumerate a dictionary D of possible passwords, in order of likelihood

```
BruteForce1(M,C):
R || C' = C
foreach pw* in D do
    C* = Enc(pw*,M,R)
    If C* = C' then
        Return pw*
```

```
BruteForce2(C):
foreach pw* in D do
    M* = Dec(pw*,C)
    If M* "looks right" then
        Return (pw*,M*)
```

PBKDF design attempts to slow down brute-force attacks



Iterating c times should slow down attacks by factor of c

Salts:

- Different derived keys, even if same password
- Slows down attacks against multiple users
- Prevents precomputation attacks, if salts chosen correctly

```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed sha1
Doing sha1 for 3s on 16 size blocks: 4109047 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 3108267 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 1755265 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 636540 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 93850 sha1's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

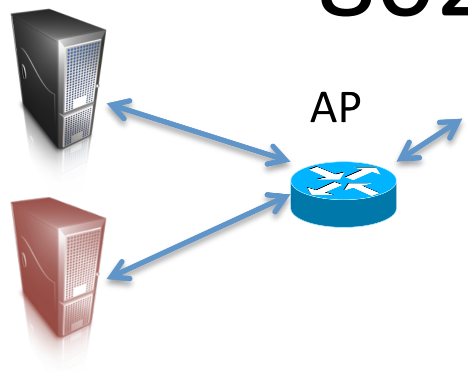
```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed aes-128-
cbc
Doing aes-128 cbc for 3s on 16 size blocks: 27022606 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 6828856 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 1653364 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 438909 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 8192 size blocks: 54108 aes-128 cbc's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

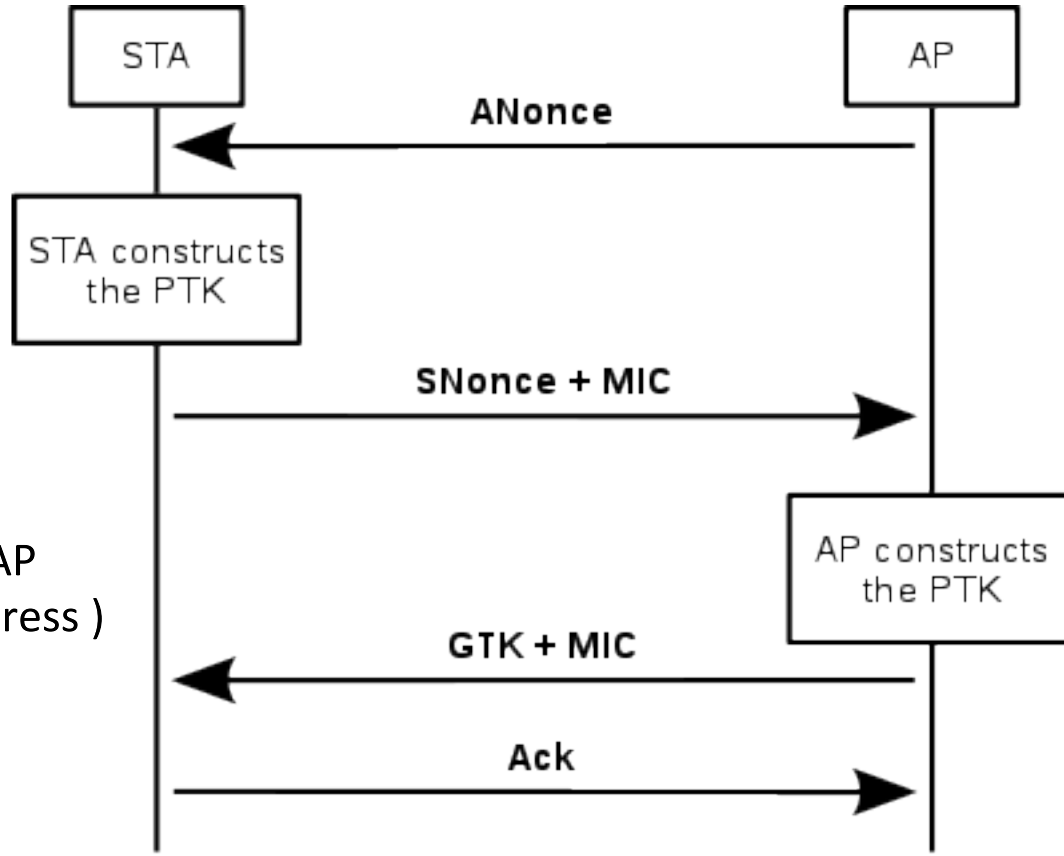
Say c = 4096. Generous back of envelope* suggests that in 1 second, can test 252 passwords and so a naïve brute-force:

6 numerical digits	10^6 = 1,000,000	~ 3968 seconds
6 lower case alphanumeric digits	36^6 = 2,176,782,336	~ 99 days
8 alphanumeric + 10 special symbols	72^8 = 722,204,136,308,736	~ 33million days

* I did the arithmetic…

802.11 WPA passwords





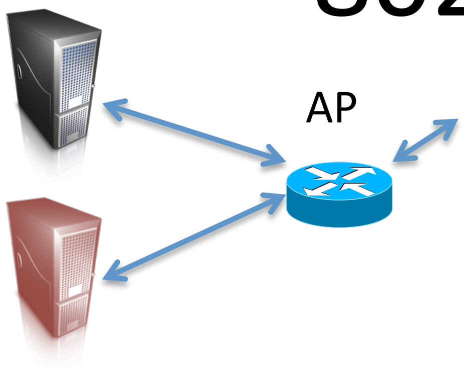
PMK = PBKDF(pw, ssid||ssidlength)
with c = 4096

PTK = H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)

MIC = HMAC-MD5(PTK, 2nd message)

So after sniffing one handshake by another party, we can mount ***offline*** brute force attack

802.11 WPA passwords



PMK = PBKDF(pw, ssid||ssidlength)
with c = 4096

PTK = H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)

MIC = HMAC-MD5(PTK, 2nd message)

<u>BruteForce(MIC,ANonce,SNonce,2nd message):</u>

```
foreach pw* in D do
    PMK* = PBKDF(pw*,ssid||ssidlength)
    PTK* = H(PMK* ||ANonce || … )
    MIC* = HMAC-MD5(PTK*, 2nd message)
    If MIC* = MIC then
        Return pw*
```

We can also use precomputation for common SSID's

PMK = F(pw,ssid)

PMK = PBKDF(pw, ssid||ssidlength)
with c = 4096

MIC = G(PMK,data)

PTK =
H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)

MIC = HMAC-MD5(PTK, 2nd message)

```
Offline(D,SsidList):
foreach pw* in D do
    foreach ssid* in Ssidlist do
        PMK* = F(pw*,ssid*)
        T[PMK*] = pw*
        Add PMK* to P[ssid*]
Return P,T
```

```
Online(P,T,MIC,ANonce, …):
foreach PMK* in P[ssid] do
    MIC* = G(PMK*,data)
    If MIC* = MIC then
        Return T[PMK*]
```

Time-space trade-off

Password recap

- Short passwords can be cracked easily
  - See also: JohnTheRipper, aircrack, tools
- Salting and iteration are helpful and needed
  - Salts must be sufficiently large and unpredictable
  - Still possible to crack in some cases

From xkcd.com