MACs, Passwords and Asymmetric encryption

CS642: Computer Security

Asymmetric encryption

MACs

Password encryption

The RSA algorithm

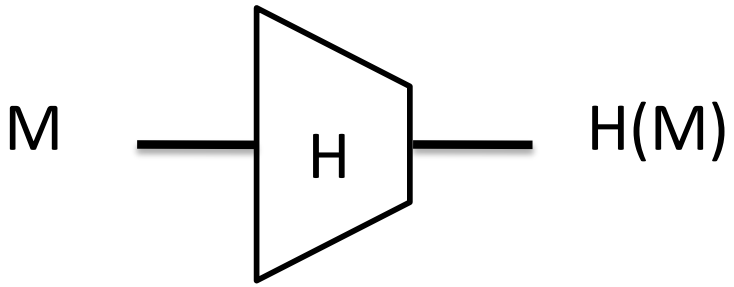
PKCS #1 encryption

Digital signing & public-key infrastructure

Hybrid encryption

Hash functions and message authentication

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



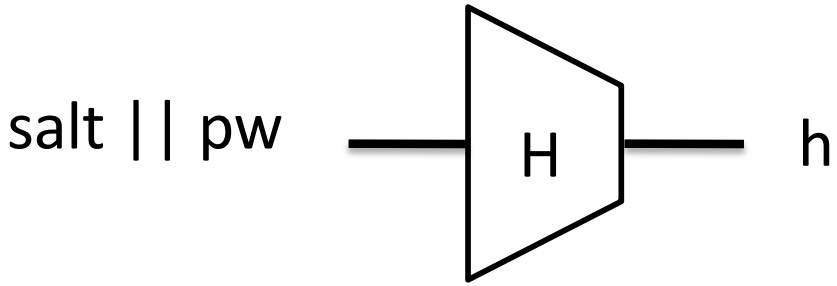
MD5: m = 128 bits
SHA-1: m = 160 bits
SHA-256: m = 256 bits

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:

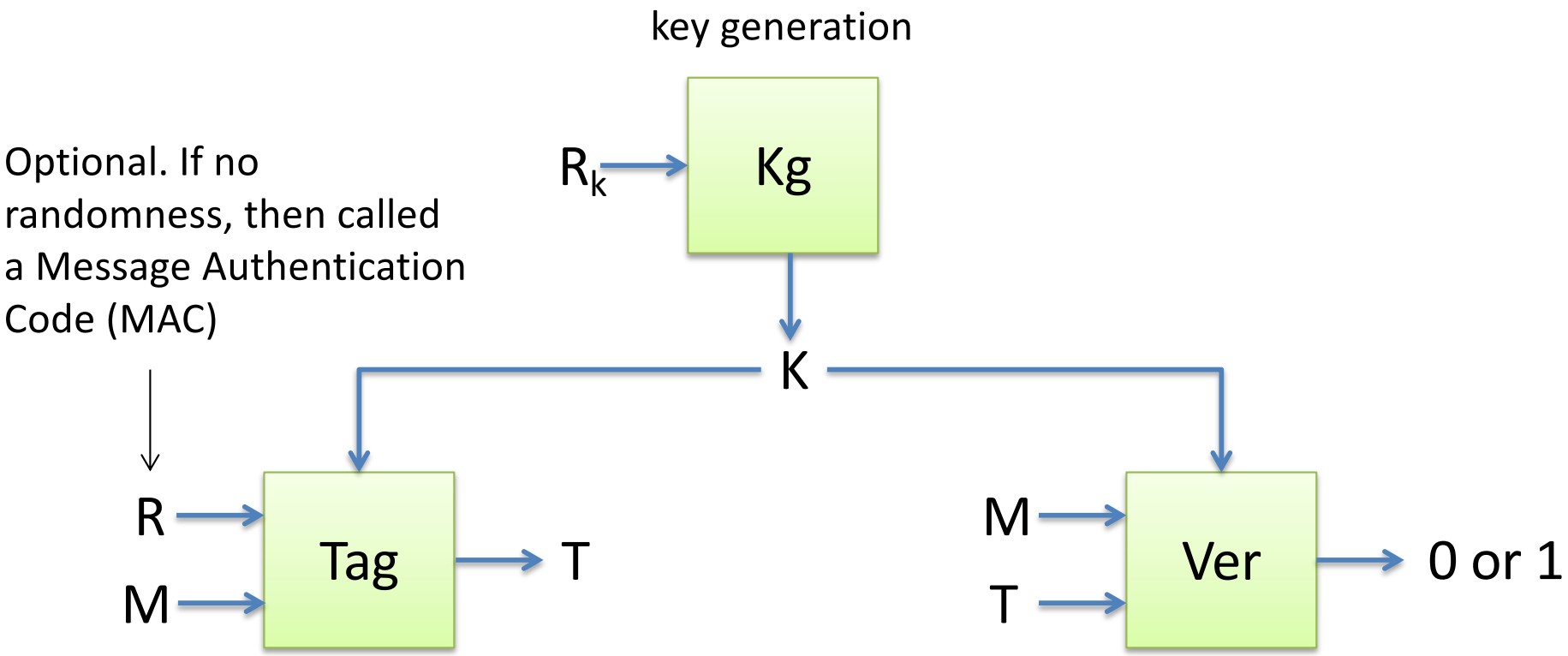


The idea: Attacker, given (salt,h), should not be able to recover pw

Or can they?

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

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

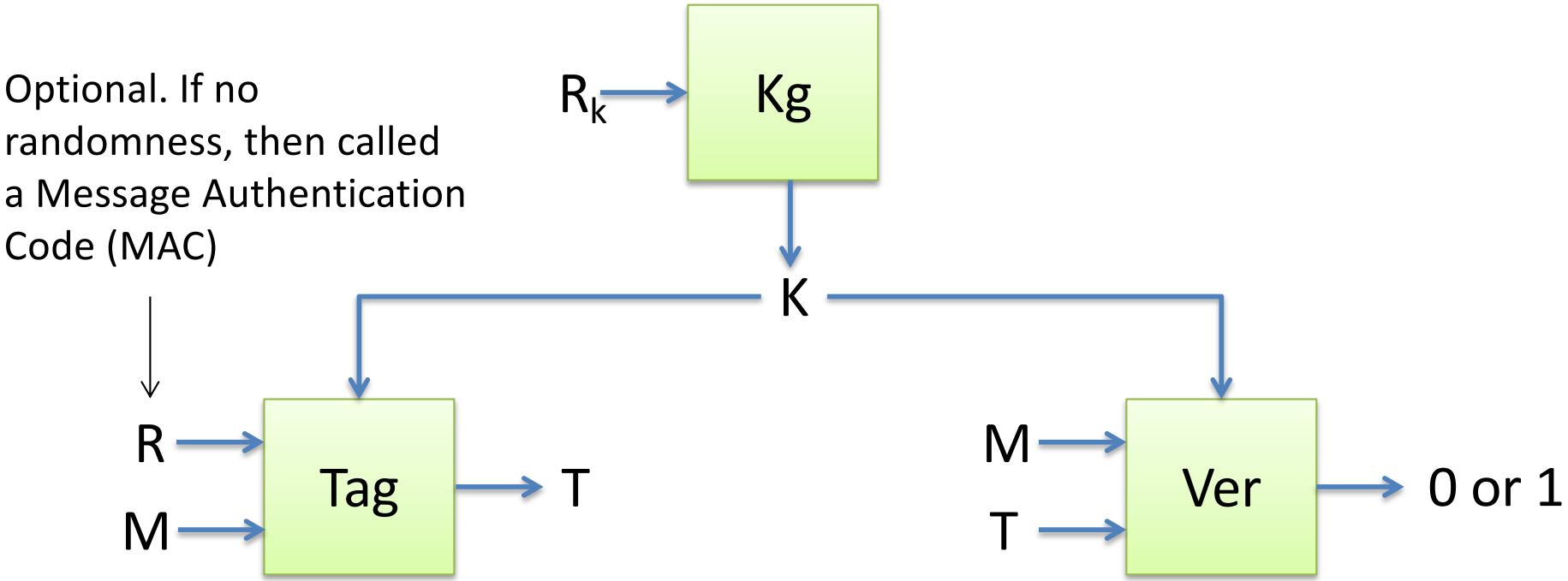
$F(K,M_1), F(K,M_2), \ldots, F(K,M_q)$ $\quad$ $U_1, U_2, \ldots, U_q$

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

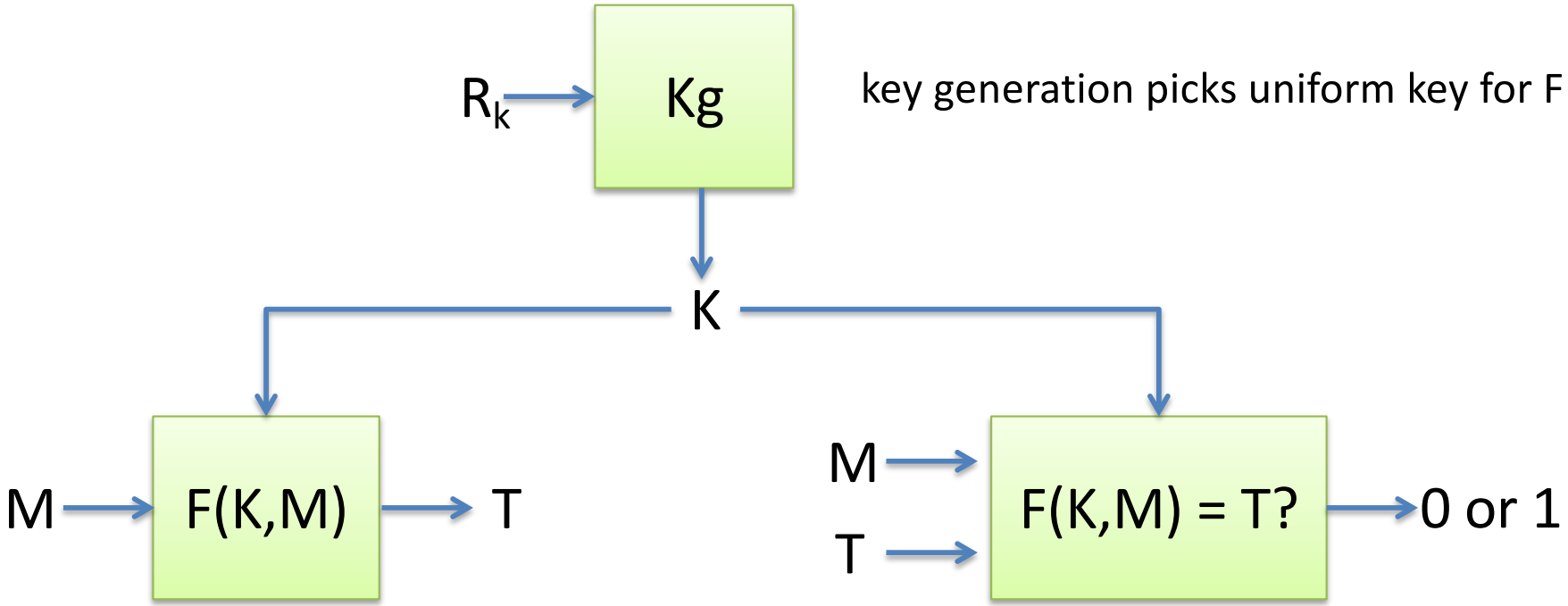
Any PRF is a good MAC



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

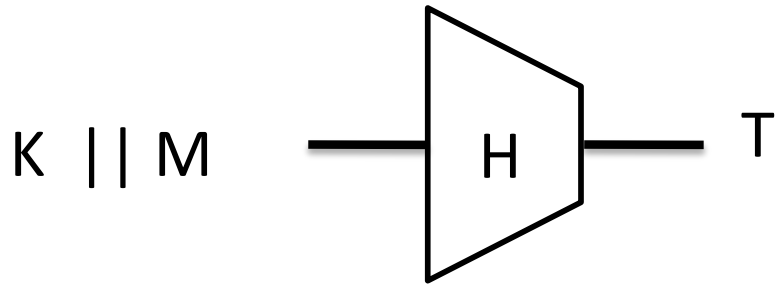


How do we instantiate F?

Attempt 1

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

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



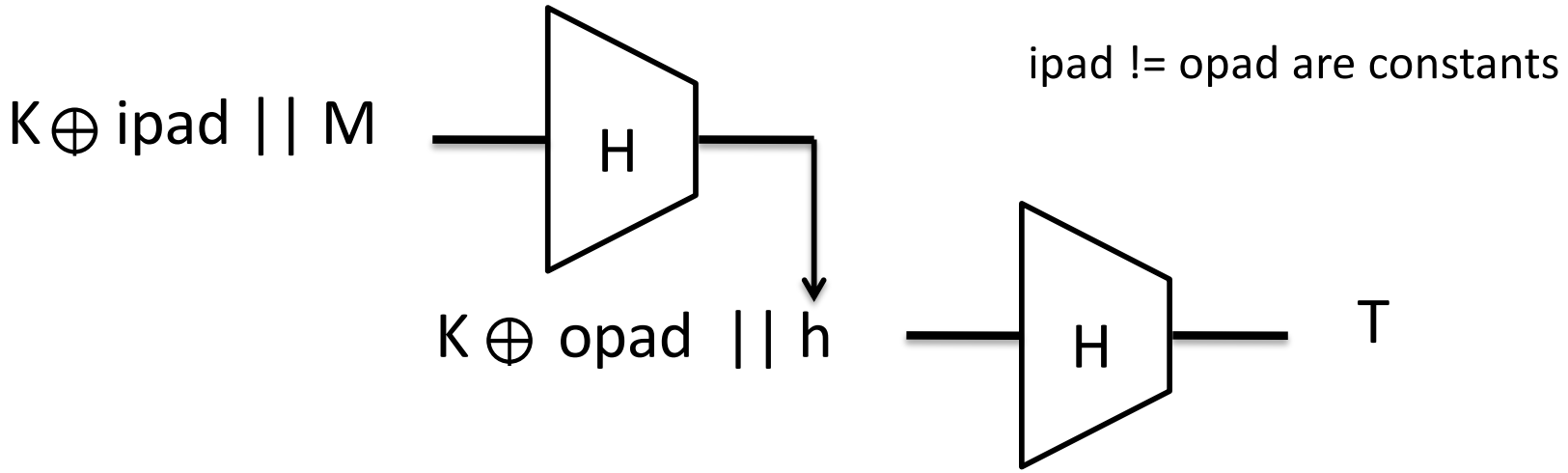
To verify a M,T pair, check if HMAC(K,M) = T

But: what if I want to append: HMACK(K, M||M') by continuing hash

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:



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

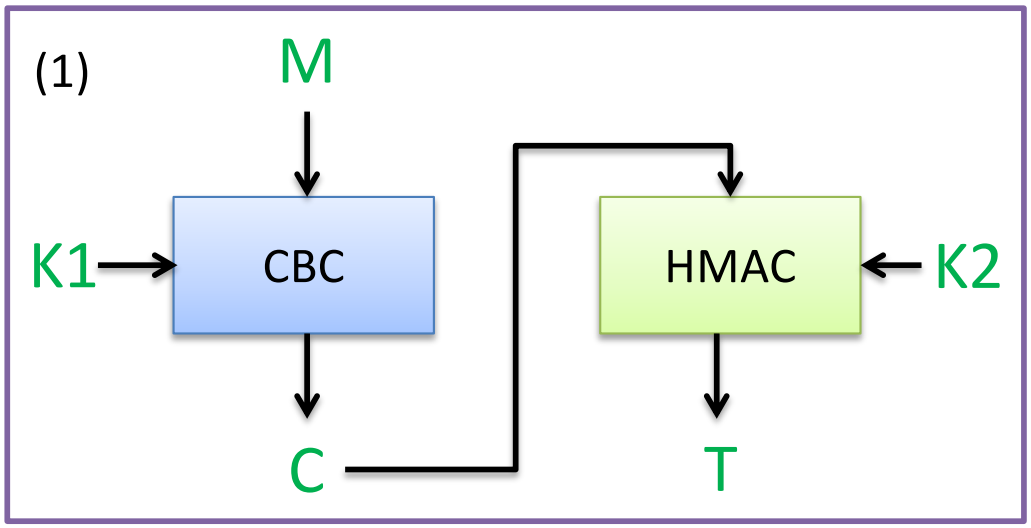
Kg outputs CBC key K1 and HMAC key K2

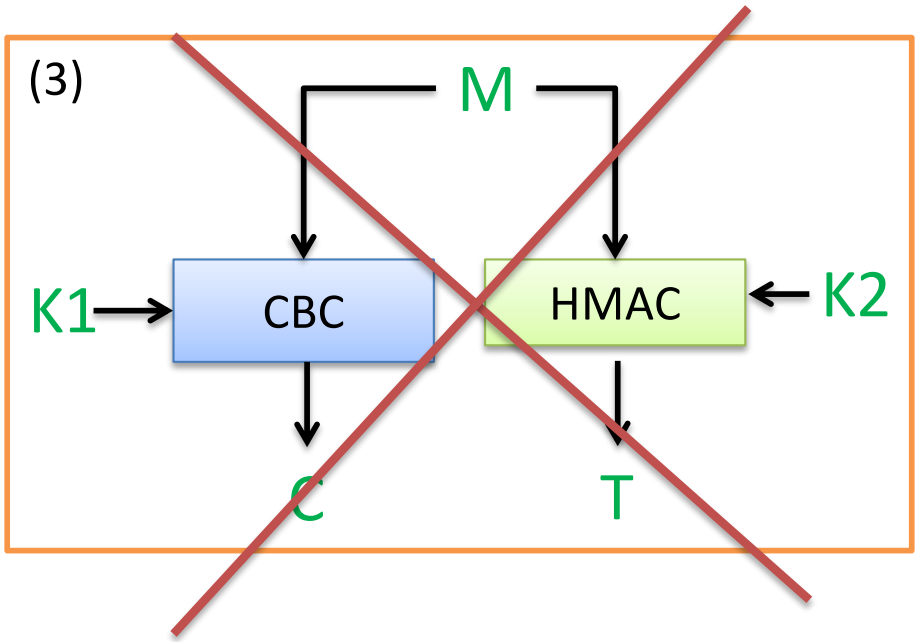
Several ways to combine:

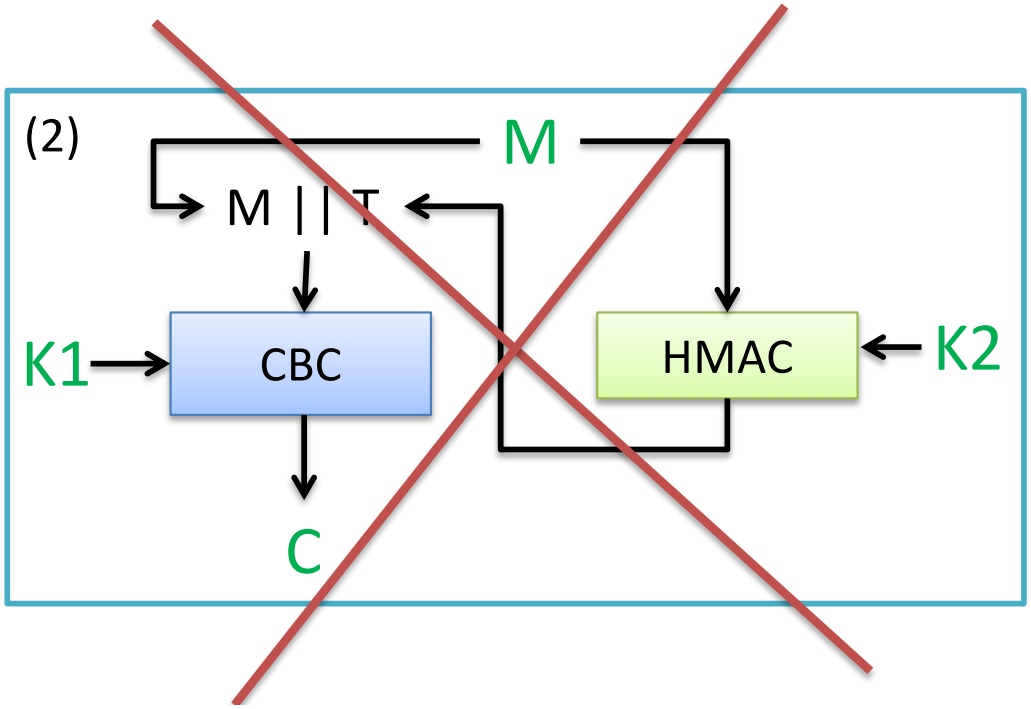
(1) encrypt-then-mac

(2) mac-then-encrypt

(3) encrypt-and-mac

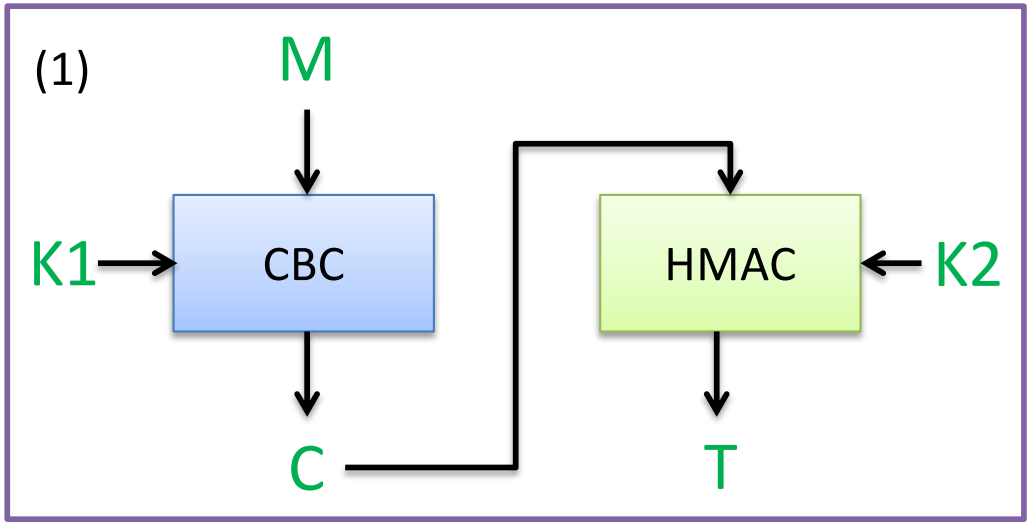






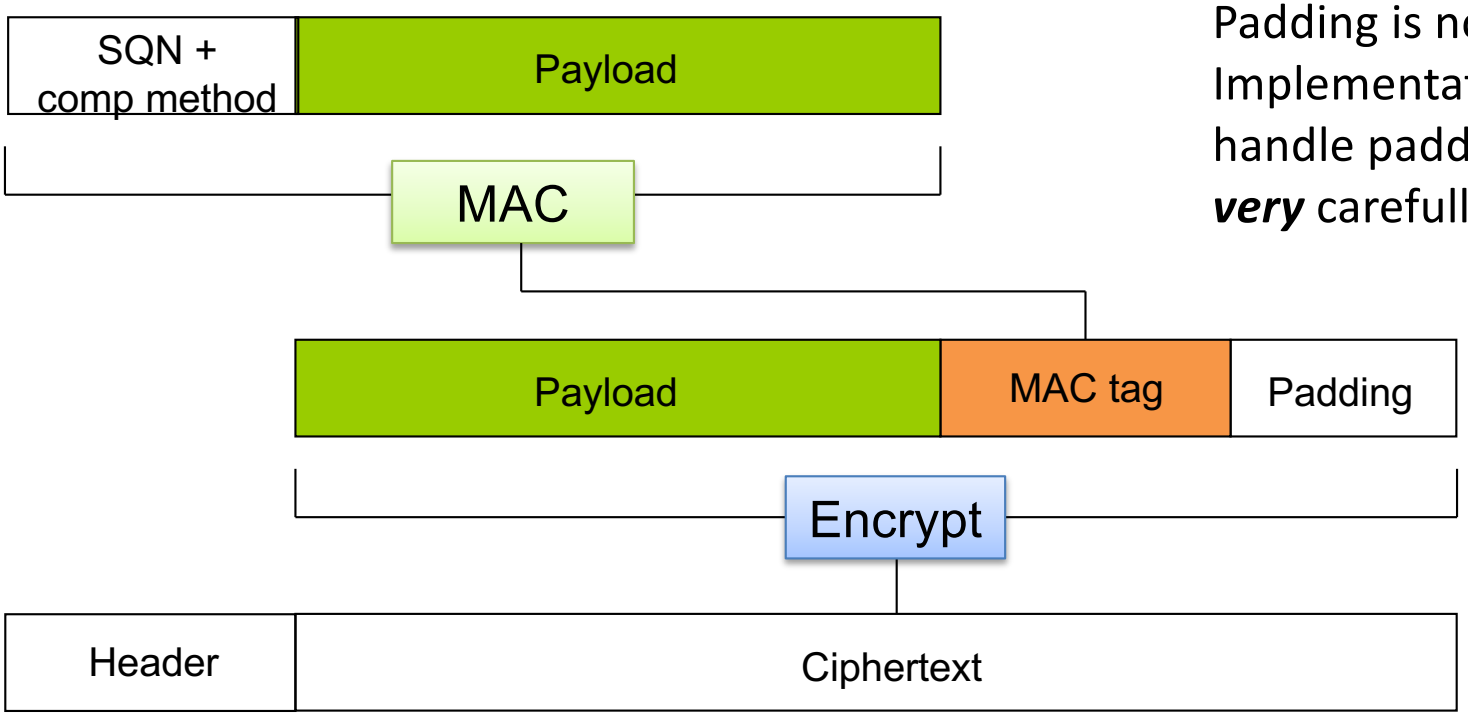
Build a new scheme from CBC and HMAC
Kg outputs CBC key K1 and HMAC key K2

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

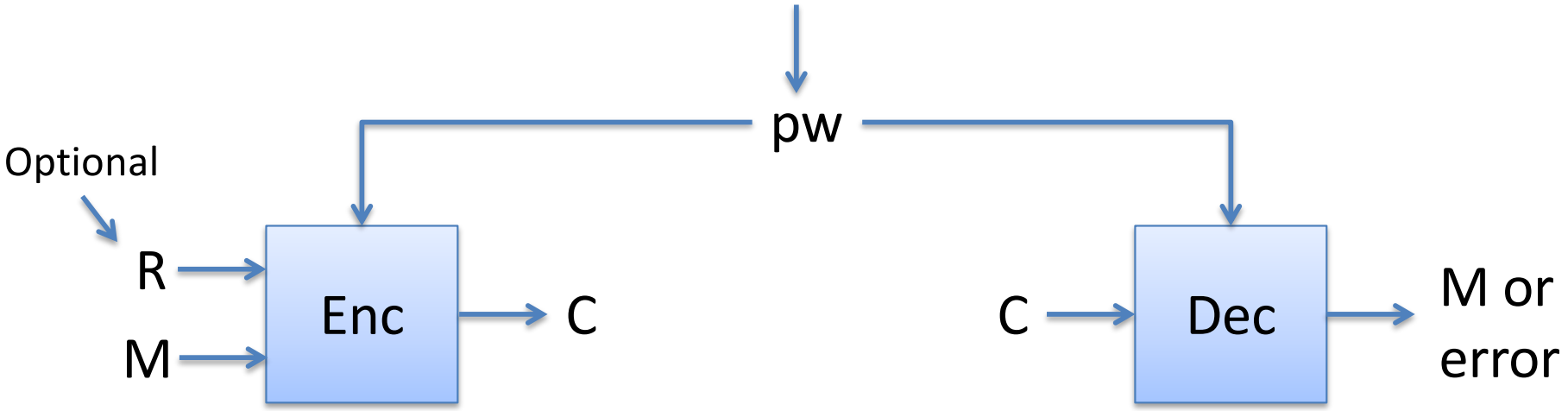
Never use CTR mode or CBC mode by themselves

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

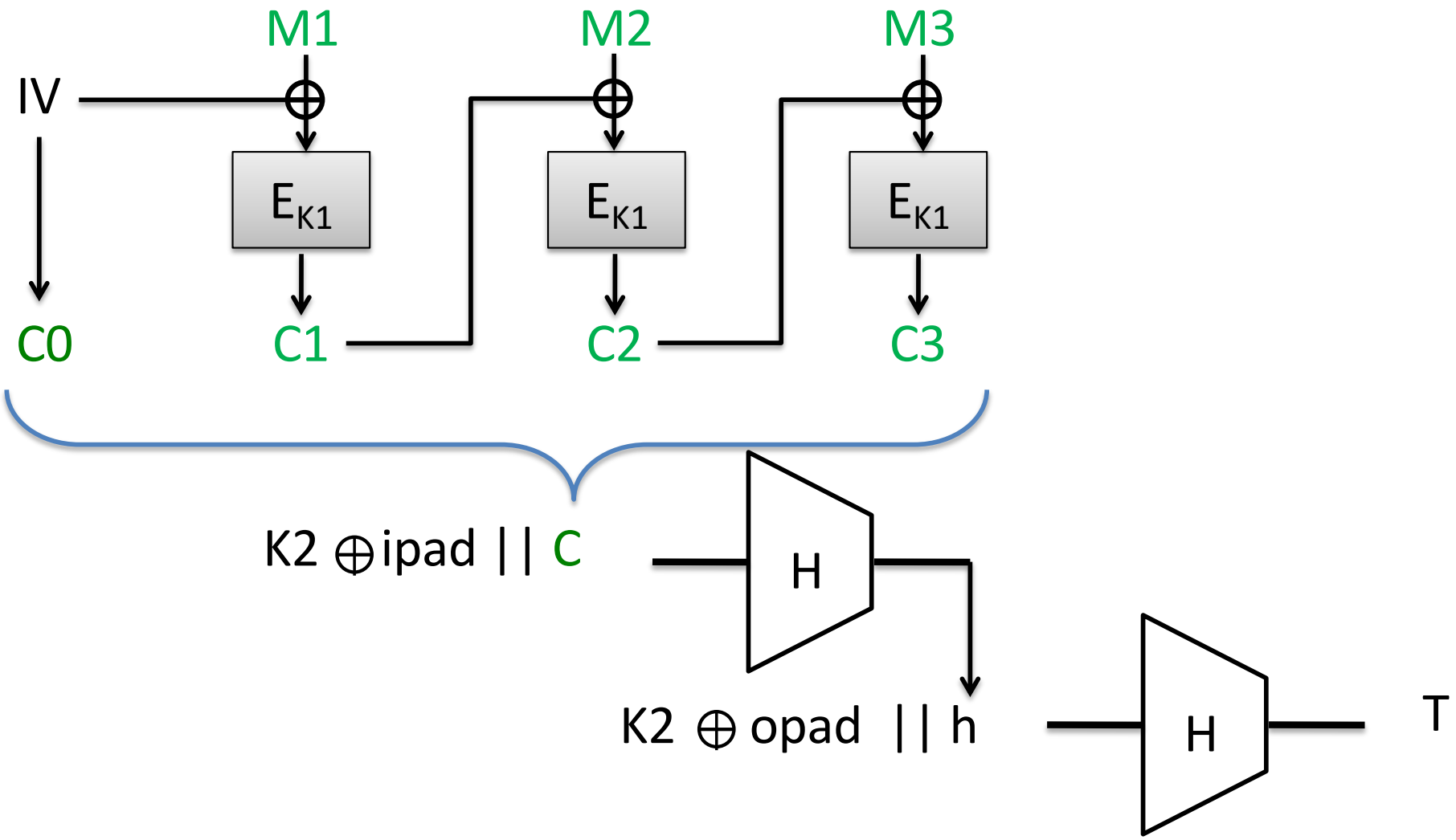
Password-based symmetric encryption



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

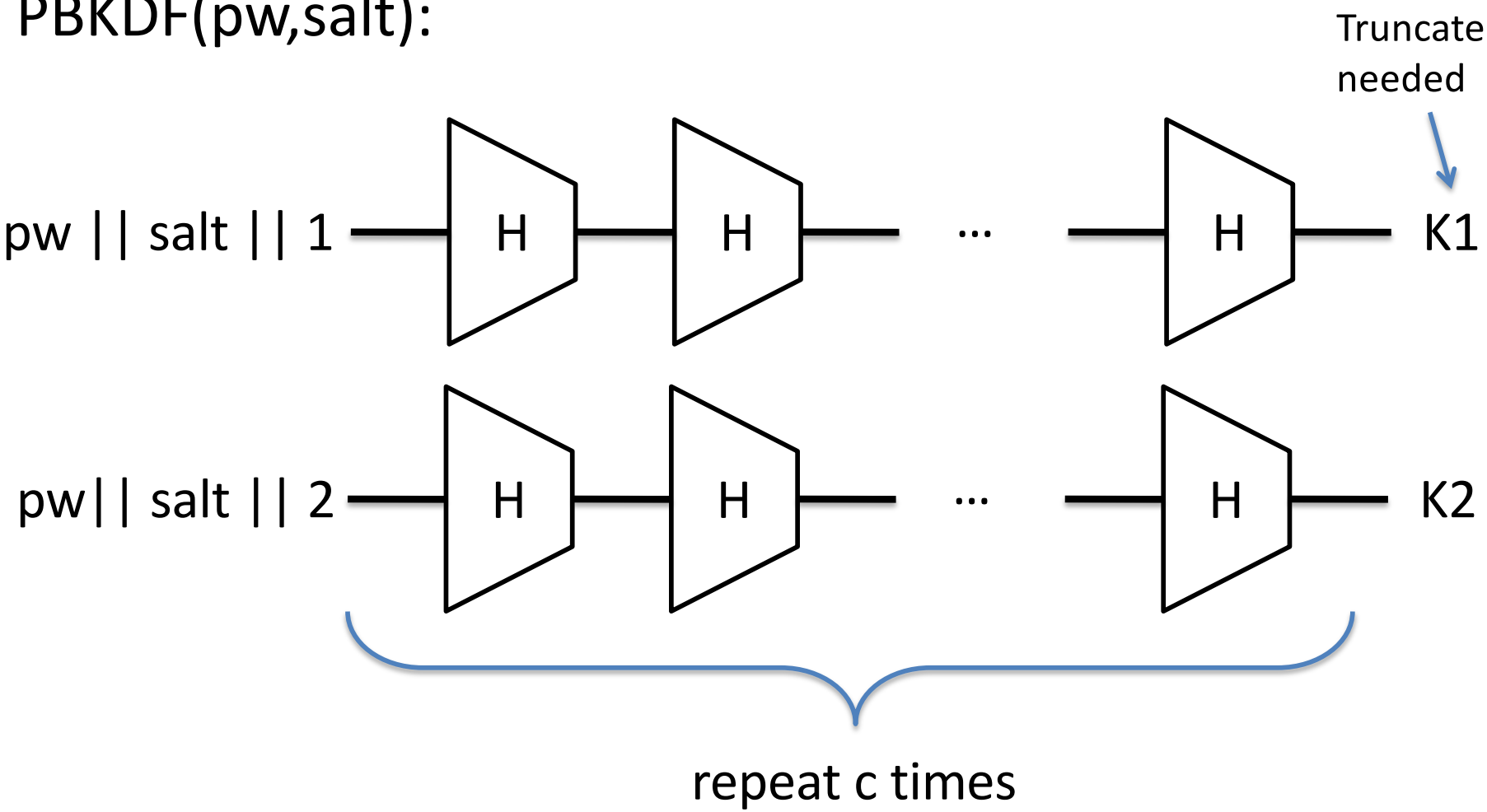


Ciphertext is C,T

How do we use with a pw?

Password-based Key Deriviation (PBKDF)

PBKDF(pw,salt):



PBKDF + Symmetric encryption yields PW-based encryption

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

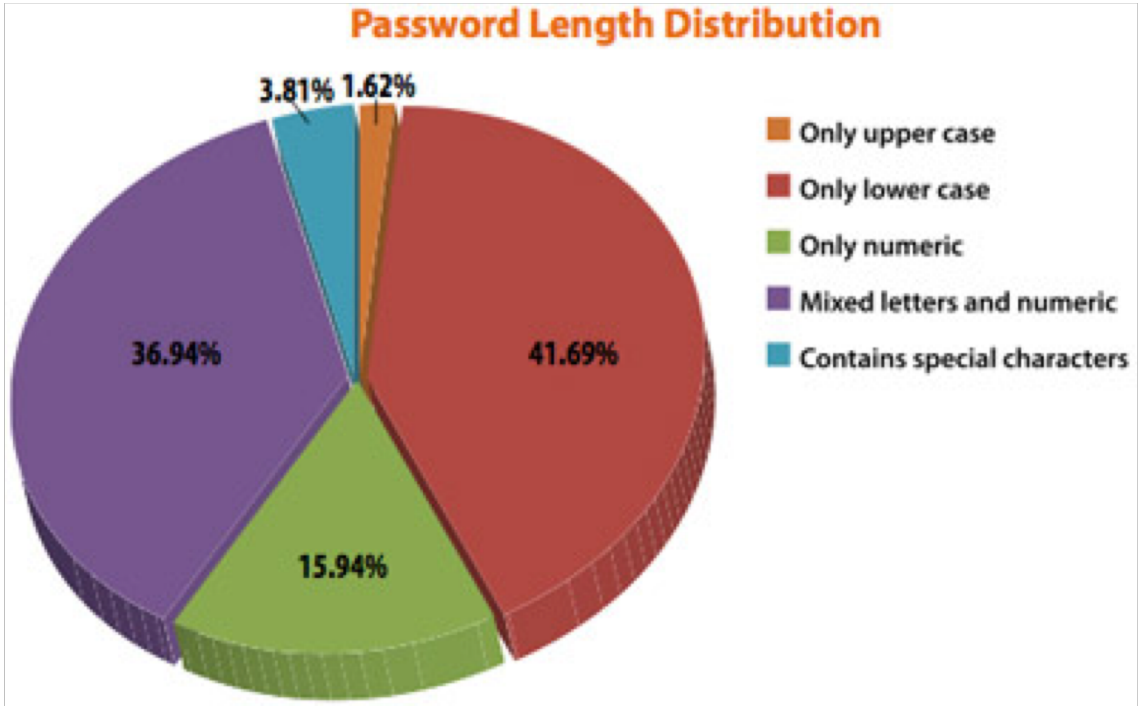
Dec(pw,C):
salt || C' = C
K = PBKDF(pw,salt)
M = Enc'(K,C')
Return M

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



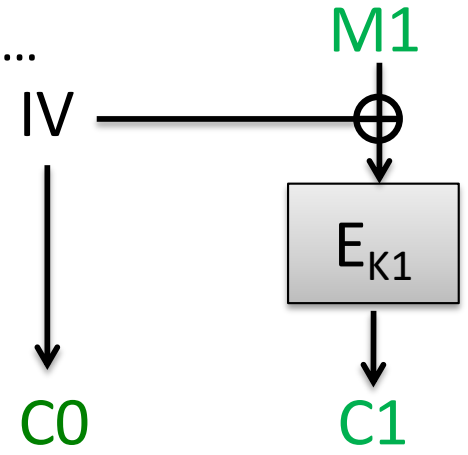
From an Imperva study of released RockMe.com password database 2010

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

BruteForce1(M,C):
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 ||…



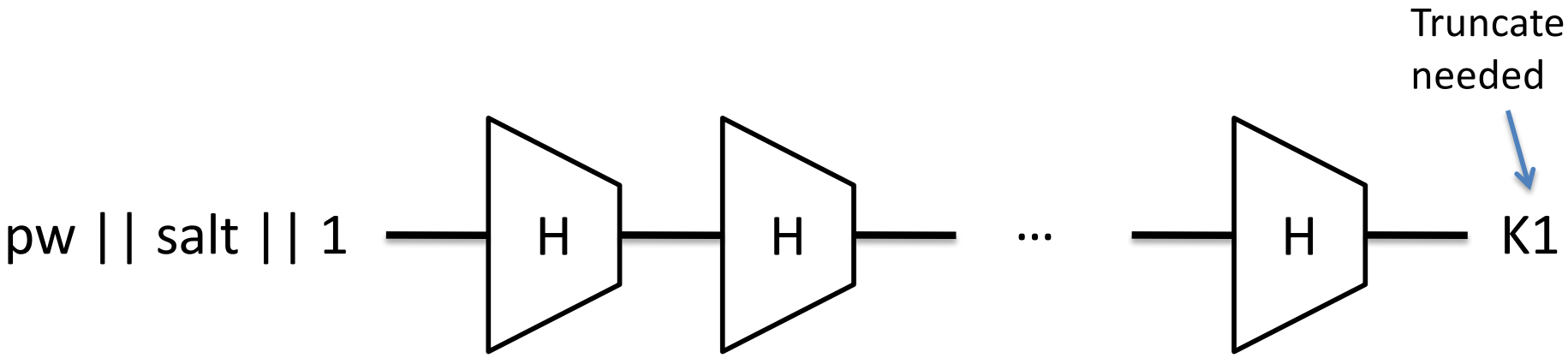
Brute-force attacks

- 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…

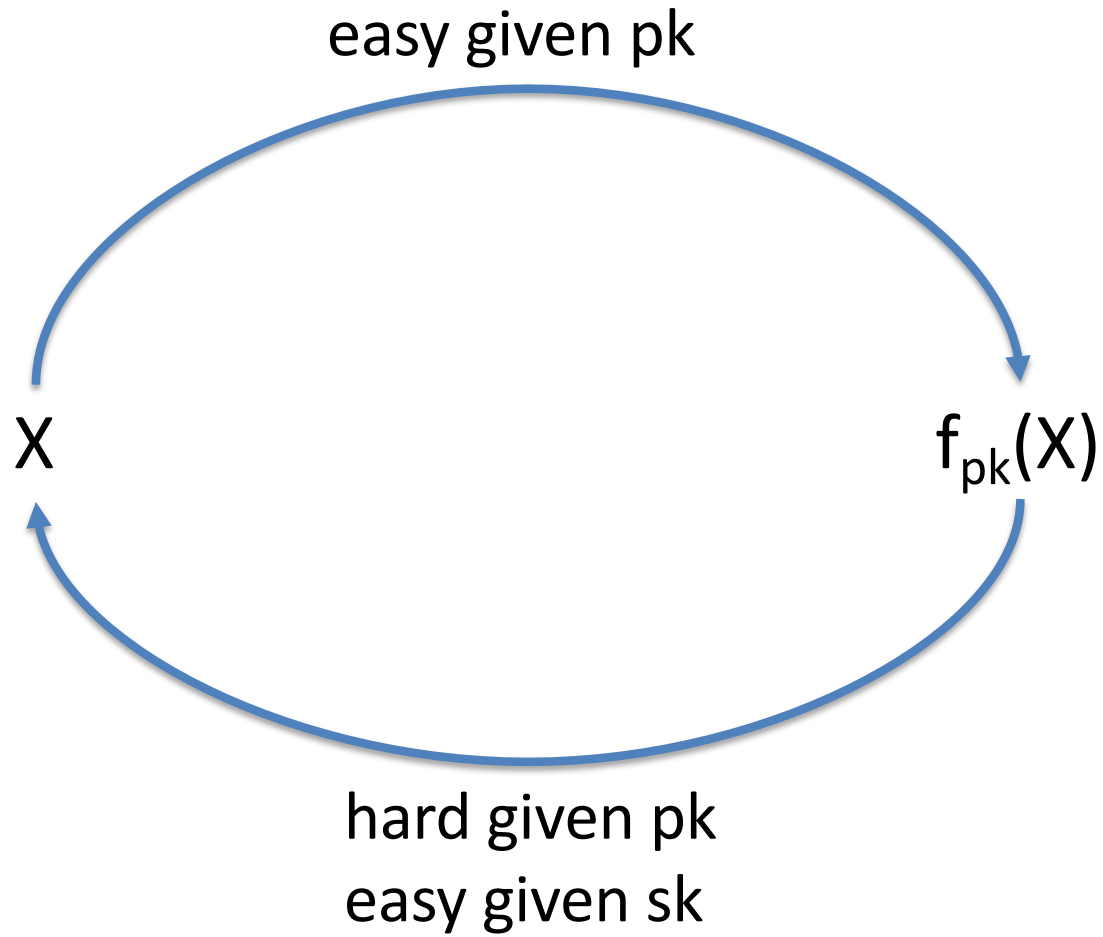
Password recap

- 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

Asymmetric Encryption

- Idea: trapdoor function
  - Easy to compute in one direction
  - Difficult to compute in opposite direction without knowledge
- Example: padlock
  - Easy to lock without key
  - Hard to open
- Other examples: Large composite numbers
  - Easy to multiple
  - Hard to factor

Asymmetric Encryption: Trapdoor function



The RSA trapdoor function

- Rivest, Shamir, Adleman 1978
- Garnered them a Turing award

RSA math

p and q be large prime numbers
N = pq
N is called the modulus

p = 7, q = 13, gives N = 91

p = 17, q = 53, gives N = 901

RSA math

p and q be large prime numbers

N = pq

N is called the modulus

$\mathbf{Z}_N = \{0,1,2,3,\ldots, N-1\}$

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

The size of a set S is denoted by |S|

gcd(X,Y) = 1 if greatest common divisor of X,Y is 1

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

N = 13 $\quad \mathbf{Z}_{13}^* = \{ 1,2,3,4,5,6,7,8,9,10,11,12 \}$

N = 15 $\quad \mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$

Def. $\phi(N) = |\mathbf{Z}_N^*|$ (This is Euler's totient function)

$\phi(13) = 12$

$\phi(15) = 8$

$\mathbf{Z}_{\phi(15)}^* = \mathbf{Z}_8^* = \{ 1,3,5,7 \}$

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

$\mathbf{Z}_N^*$ is a group under modular multiplication

Fact. For any a,N with N > 0, there exists unique q,r such that

$a = Nq + r$ and $0 \le r < N$

$17 \bmod 15 = 2$

Def. $a \bmod N = r \in \mathbf{Z}_N$ $105 \bmod 15 = 0$

Def. $a \equiv b \pmod N$ iff $(a \bmod N) = (b \bmod N)$

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

$\mathbf{Z}_N^*$ is a group under modular multiplication

$$\mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$$

$2 \bullet 7 \equiv 14 \pmod{15}$

$4 \bullet 8 \equiv 2 \pmod{15}$

Closure: for any $a,b \in \mathbf{Z}_N^*$ $\quad a \bullet b \bmod N \in \mathbf{Z}_N^*$

Def. $a^i \bmod N = \underbrace{a \bullet a \bullet a \bullet \ldots \bullet a}_{i \text{ times}} \bmod N$

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

Claim: Suppose $e,d \in \mathbf{Z}^*_{\phi(N)}$ satisfying $ed \bmod \phi(N) = 1$ then for any $x \in \mathbf{Z}_N^*$ we have that

$$(x^e)^d \bmod N = x$$

$$(x^e)^d \bmod N = x^{(ed \bmod \phi(N))} \bmod N$$
$$= x^1 \bmod N$$
$$= x \bmod N$$

First equality is by Euler's Theorem

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

Claim: Suppose $e,d \in \mathbf{Z}_{\phi(N)}^*$ satisfying $ed \bmod \phi(N) = 1$ then for any $x \in \mathbf{Z}_N^*$ we have that

$$(x^e)^d \bmod N = x$$

$\mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$ $\mathbf{Z}_{\phi(15)}^* = \{ 1,3,5,7 \}$

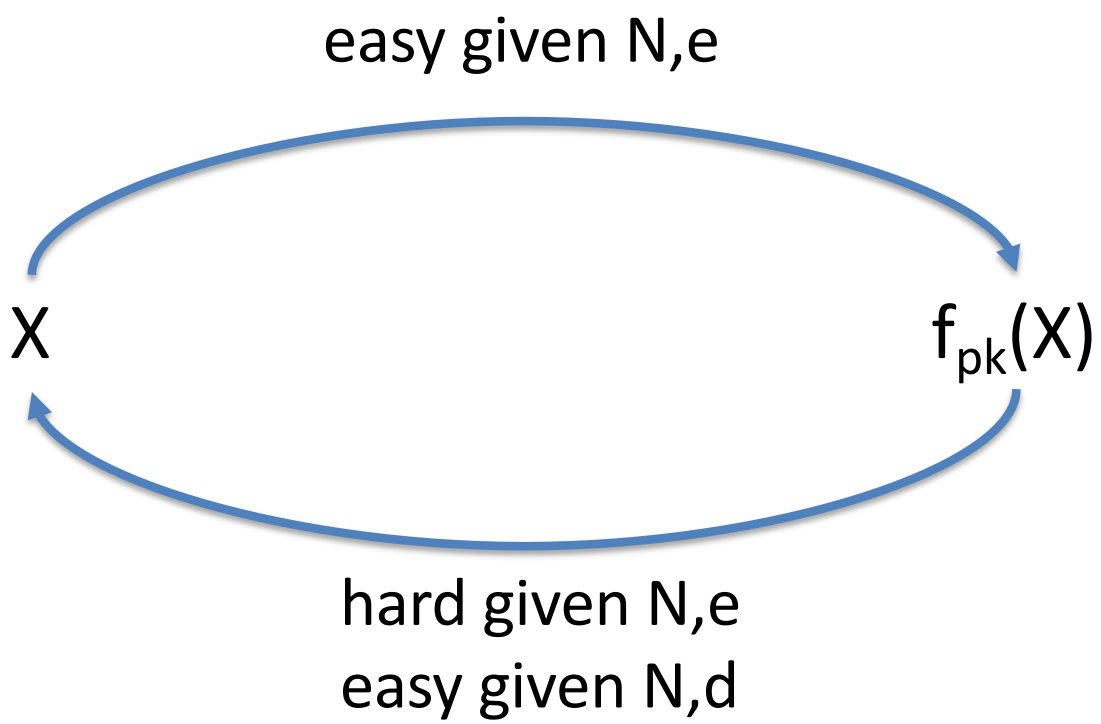
$e = 3$, $d = 3$ gives $ed \bmod 8 = 1$

x	1	2	4	7	8	11	13	14
$x^3 \bmod 15$	1	8	4	13	2	11	7	14

The RSA trapdoor permutation

pk = (N,e) sk = (N,d) with ed mod $\phi(N) = 1$

$f_{N,e}(x) = x^e \bmod N$ $g_{N,d}(y) = y^d \bmod N$



The RSA trapdoor permutation

pk = (N,e) sk = (N,d) with $ed \bmod \phi(N) = 1$

$f_{N,e}(x) = x^e \bmod N$ $g_{N,d}(y) = y^d \bmod N$

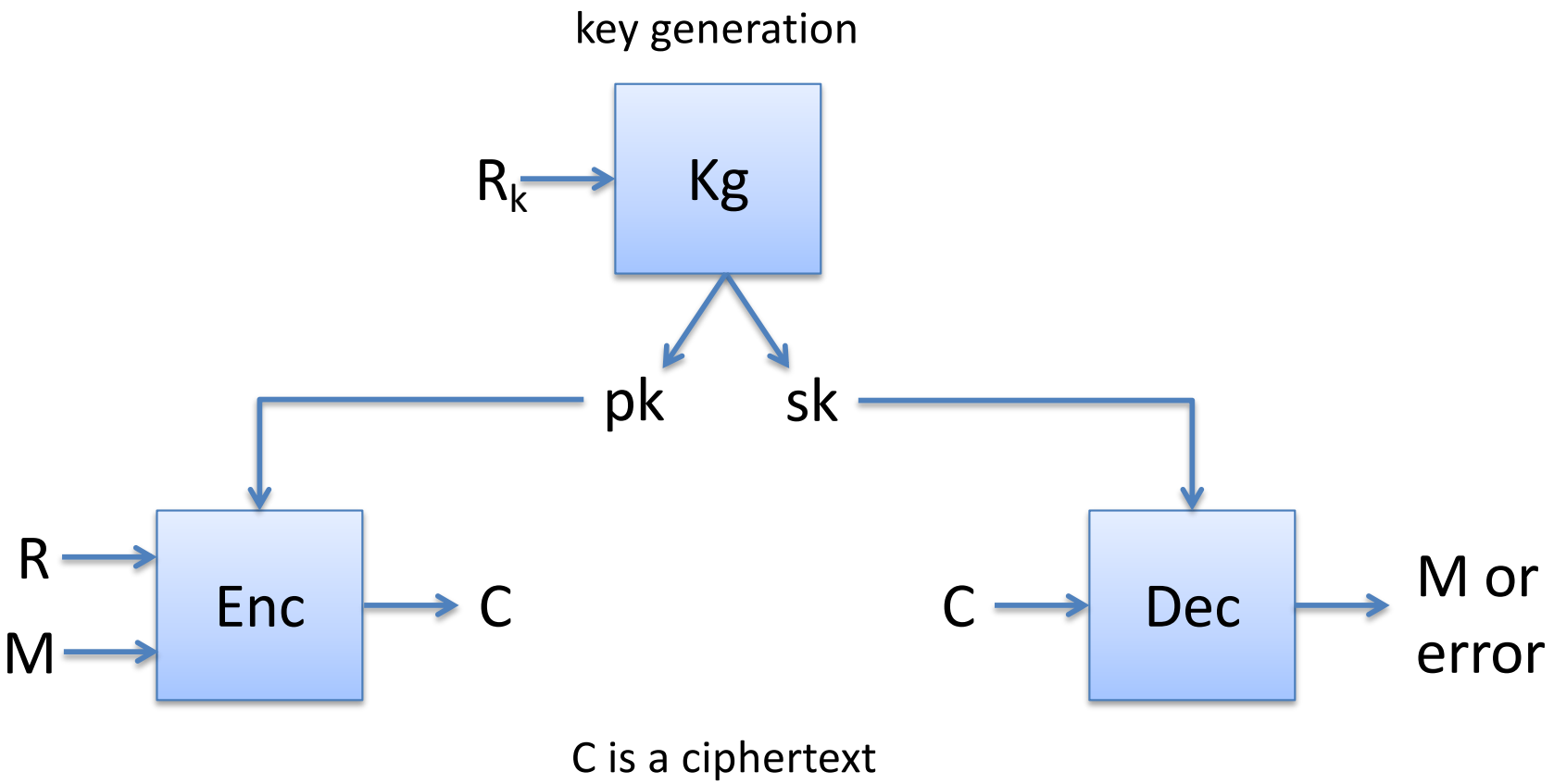
But how do we find suitable N,e,d ?

If p,q distinct primes and N = pq then $\phi(N) = (p-1)(q-1)$

Encrypt/decrypt with SK: $C = m^e \bmod N$
Encrypt/decrypt with PK: $M = c^d \bmod N$

Key generation: find large primes P, Q

Public-key encryption



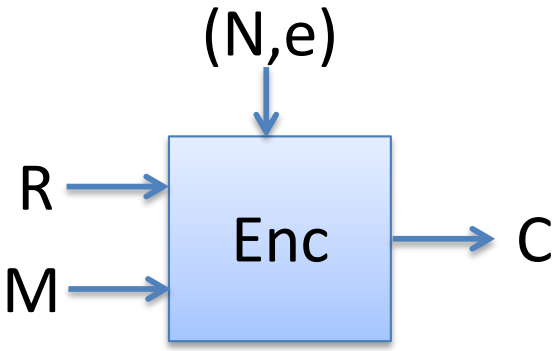
Correctness: D(sk , E(pk,M,R)) = M with probability 1 over randomness used

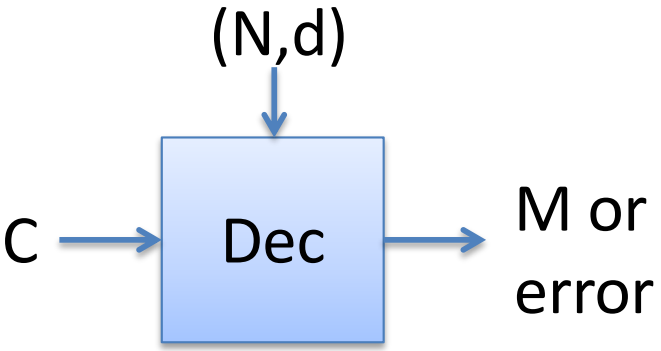
PKCS #1 RSA encryption

Kg outputs (N,e),(N,d) where $|N|_8 = n$

Let $B = \{0,1\}^8 / \{00\}$ be set of all bytes except 00

Want to encrypt messages of length $|M|_8 = m$





Enc((N,e), M, R)

pad = first n - m - 3 bytes from R that are in B

X = 00 || 02 || pad || 00 || M

Return X^e mod N

Dec((N,d), C)

X = C^d mod N ; aa||bb||w = X

If (aa ≠ 00) or (bb ≠ 02) or (00 ∉ w)

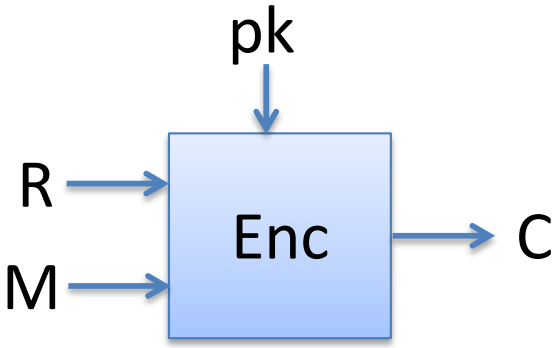
  Return error

pad || 00 || M = w

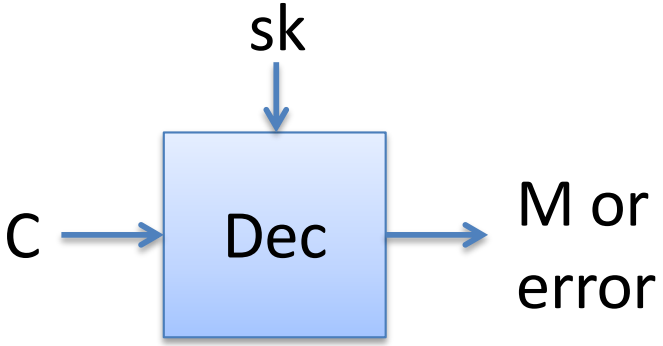
Return M

Hybrid encryption

Kg outputs (pk,sk)



Enc(pk, M, R)
K||R1||R2 = R
C1 = Enc(pk,K,R1)
C2 = Enc(K,M,R2)
Return (C1,C2)



Dec(sk, (C1,C2))
K = Dec(sk,C1)
M = Dec(K,C2)
Return M

Bank customer

TLS handshake for RSA transport

Bank

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

← CERT = (pk of bank, signature over it)

Pick random PMS
C <- E(pk,PMS)

C →

PMS <- D(sk,C)

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PS, “master secret” || Nc || Ns)

Factoring composites

- What is p,q for N = 901?

```
Factor(N):
for i = 2 , … ,  sqrt(N) do
      if N mod i = 0 then
            p = i
            q = N / p
            Return (p,q)
```

Woops… we can always factor

But not always efficiently:
Run time is sqrt(N)

$O(\text{sqrt}(N)) = O(e^{0.5 \ln(N)})$

Factoring composites

Algorithm	**Time to factor N**
Naïve	$O(e^{0.5 \ln(N)})$
Quadratic sieve (QS)	$O(e^{c})$ $c = d (\ln N)^{1/2} (\ln \ln N)^{1/2}$
Number Field Sieve (NFS)	$O(e^{c})$ $c = 1.92 (\ln N)^{1/3} (\ln \ln N)^{2/3}$

Factoring records

Algorithm	**Year**	**Algorithm**	**Time**
RSA-400	1993	QS	830 MIPS years
RSA-478	1994	QS	5000 MIPS years
RSA-515	1999	NFS	8000 MIPS years
RSA-768	2009	NFS	~2.5 years
RSA-1024	Note yet		

RSA-x is an RSA challenge modulus of size x bits

TLS handshake for RSA transport

Bank customer

Bank

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods

Pick random Ns

ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

CERT = (pk of bank, signature over it)

C

Pick random PMS
C <- E(pk,PMS)

PMS <- D(sk,C)

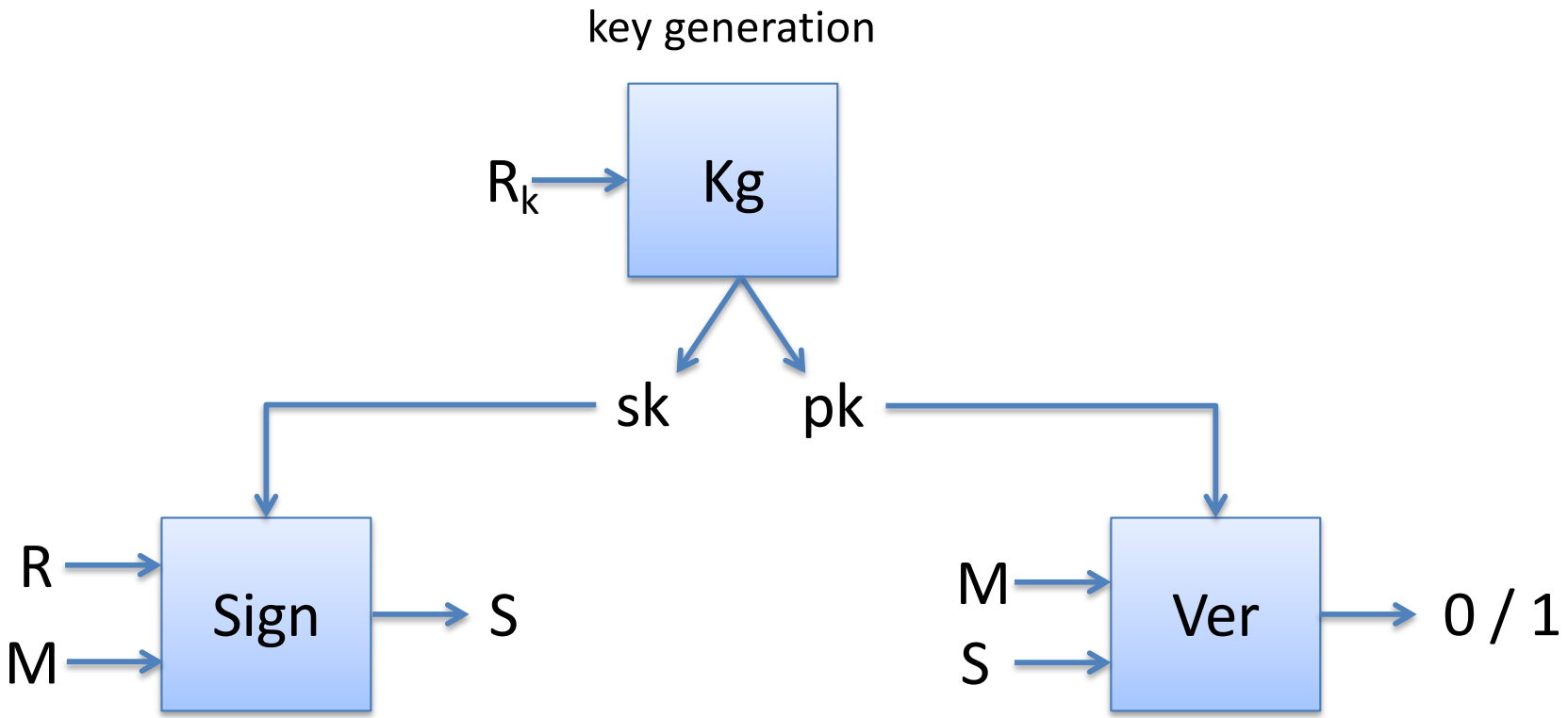
ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) }

ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) }

Bracket notation means contents encrypted

MS <- PRF(PS, “master secret” || Nc || Ns)

Digital signatures



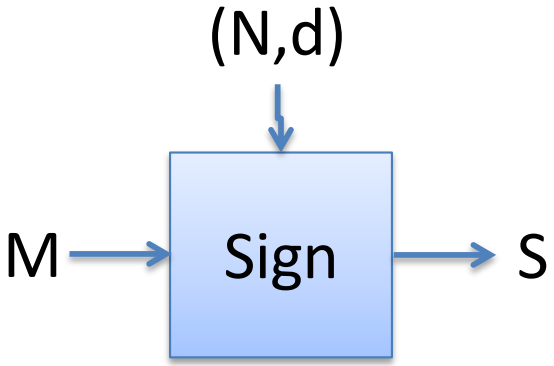
Anyone with public key can verify a signature

Only holder of secret key should be able to generate a signature

Full Domain Hash RSA

Kg outputs pk = (N,e) , sk = (N,d)

H is a hash function

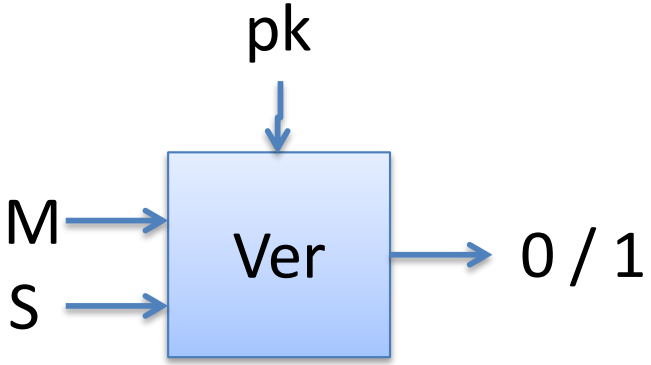


Sign((N,d), M)

$X = 00 \,||\, H(1||M) \,||\, \ldots \,||\, H(k||M)$

$S = X^d \bmod N$

Return S



Ver((N,e), M, S)

$X = S^e \bmod N$

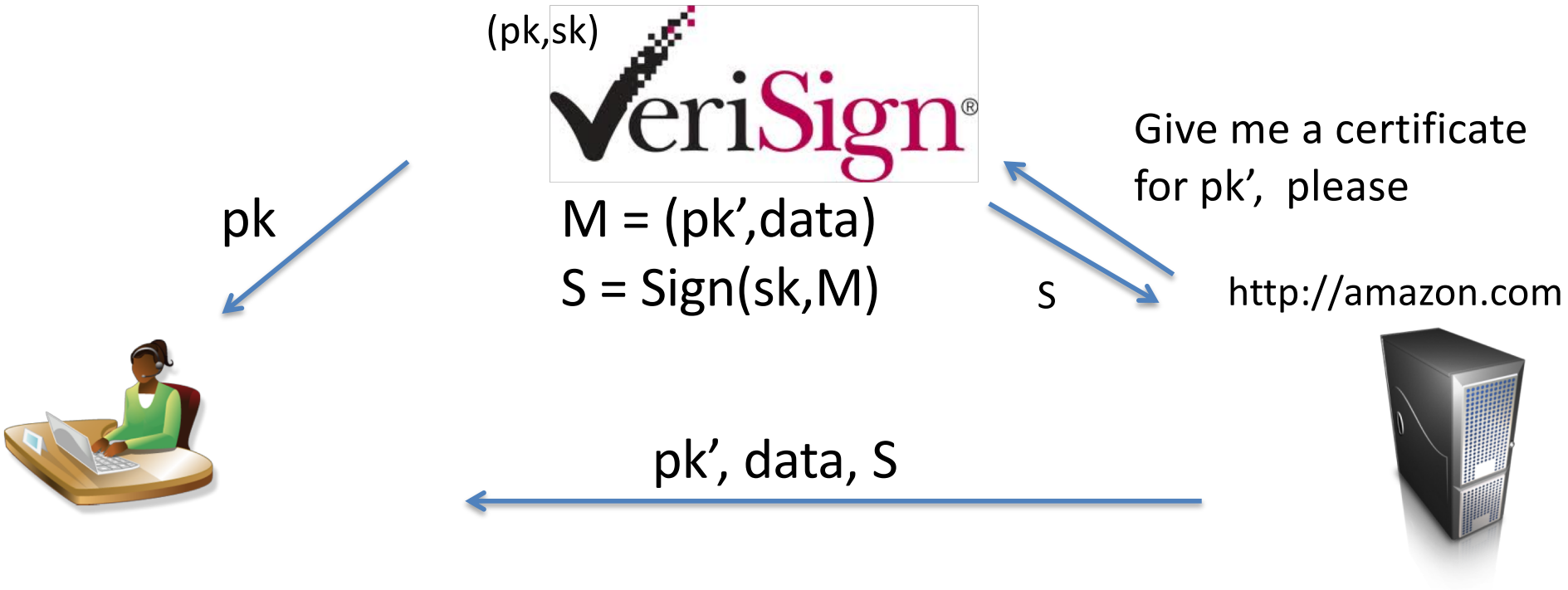
$X' = 00 \,||\, H(1||M) \,||\, \ldots \,||\, H(k||M)$

If X = X' then

    Return 1

Return 0

Certificate Authorities and Public-key Infrastructure



M = (pk',data)
If Ver(pk,M,S) then
trust pk'

This prevents man-in-the-middle (MitM) attacks