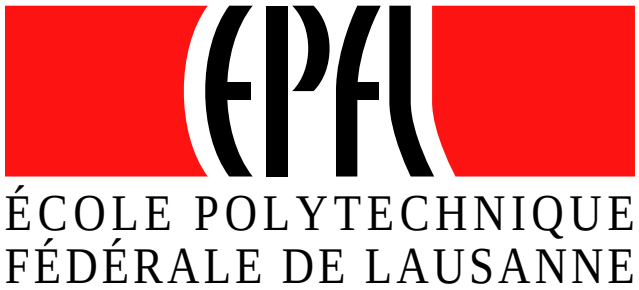
Secure Channel by SSL/TLS

A Cryptographic Maze

Serge Vaudenay



http://lasecwww.epfl.ch/

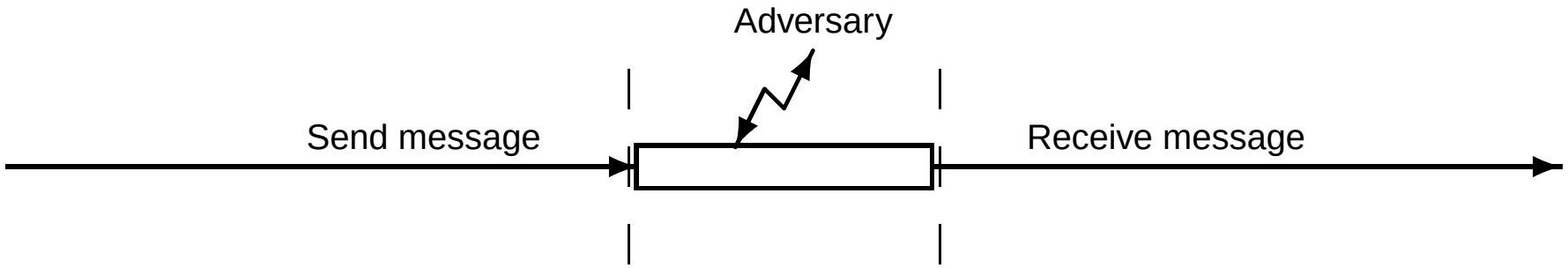
LASEC

Example of Critical Application

Requirements

- strong bidirectional authentication
- confidentiality of communications
- integrity of communication
- need not the client part to be strongly secure

Basics on Communication Security



- **Authentication**: only the legitimate sender can send
- **Integrity**: the received and sent messages must be the same
- **Confidentiality**: only the legitimate receiver can read

A Few Cryptographic Primitives

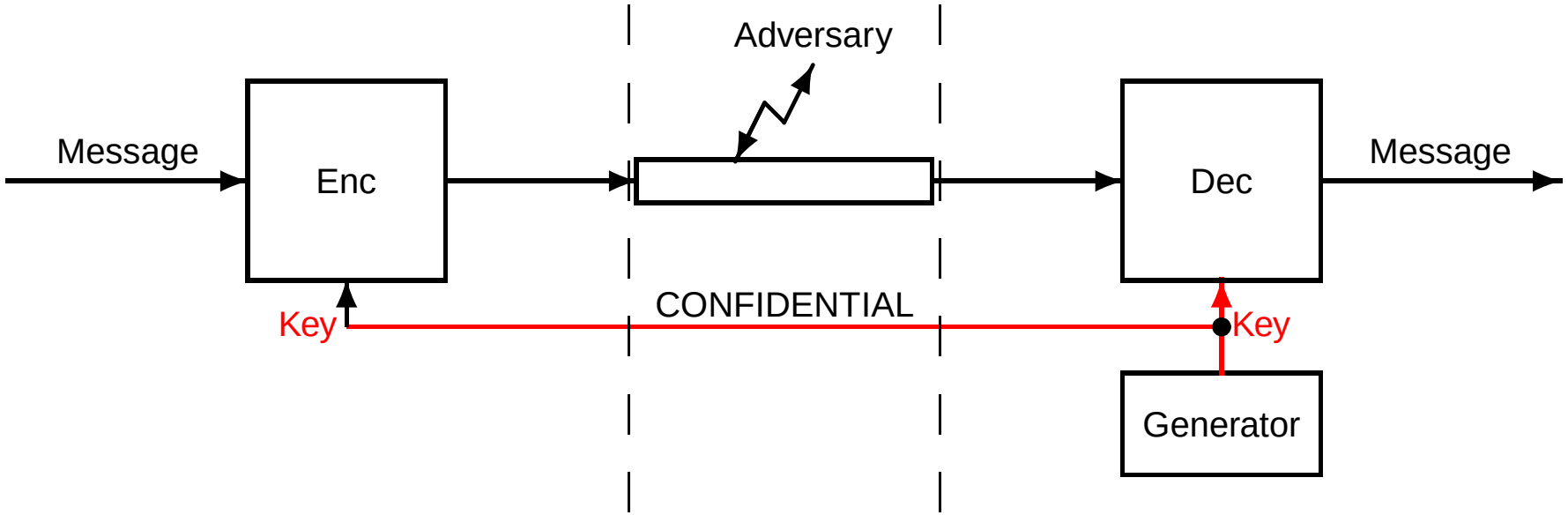
Conventional

- hash function
- symmetric encryption
- message authentication code

Asymmetric

- key agreement protocol
- public-key cryptosystem
- digital signature

Confidentiality by Symmetric Encryption



Security of Symmetric Encryption

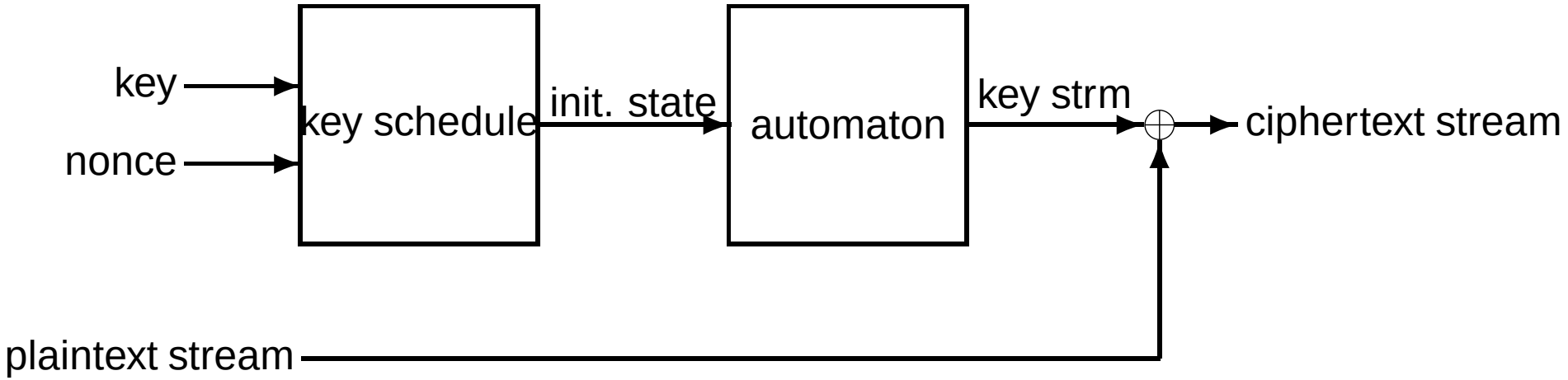
Key-recovery : it is hard to recover the secret key even if we have access to an encryption/decryption oracle

Message-recovery : after playing with an encryption/decryption oracle, it is hard to decrypt a new challenged ciphertext

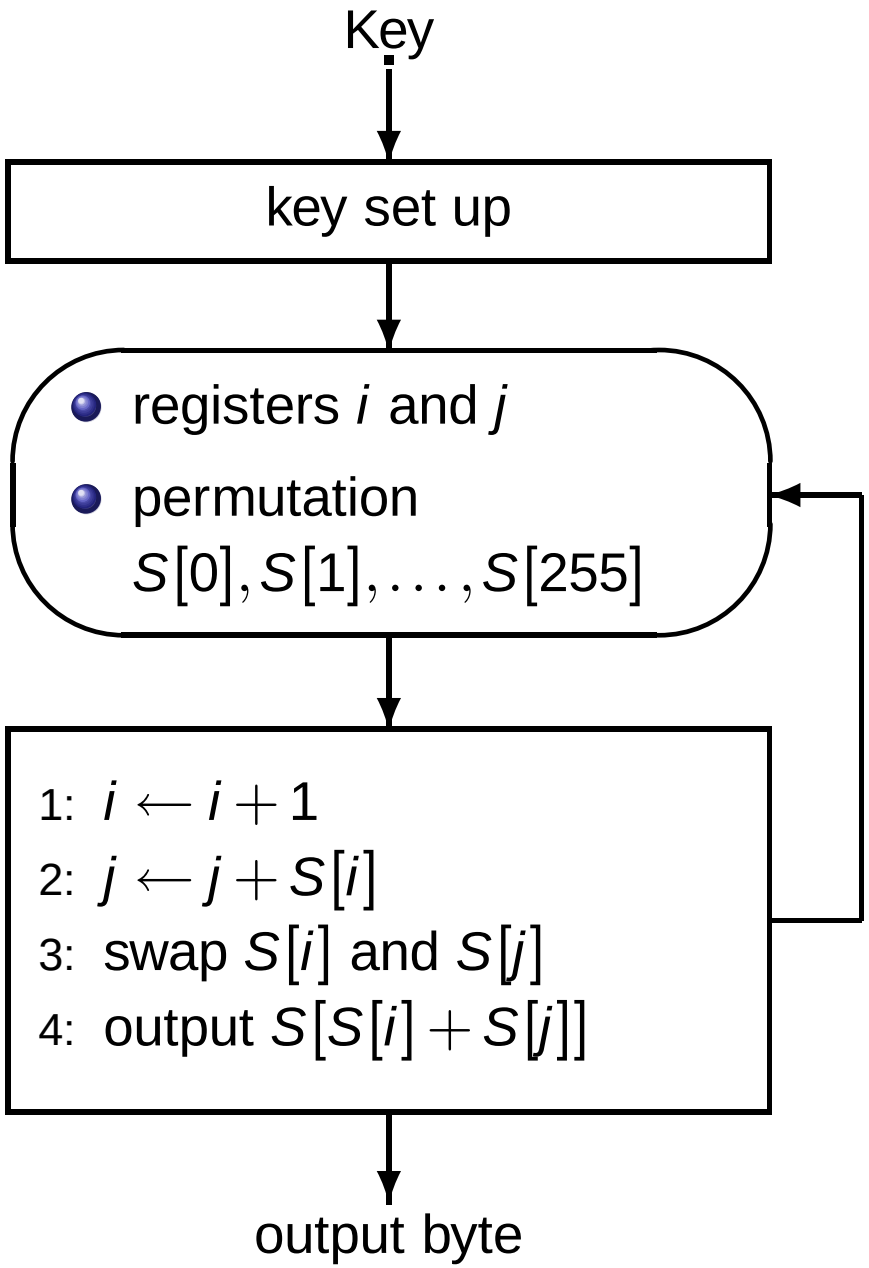
Two Categories of Symmetric Encryption

stream ciphers	block ciphers
RC4	**DES**
GSM–A5/1	**3DES**
Bluetooth–E0	IDEA
DVB-CSA	BLOWFISH
...	RC5
	AES
	KASUMI
	SAFER
	CS-Cipher
	FOX
	...

Stream Ciphers from a High Level



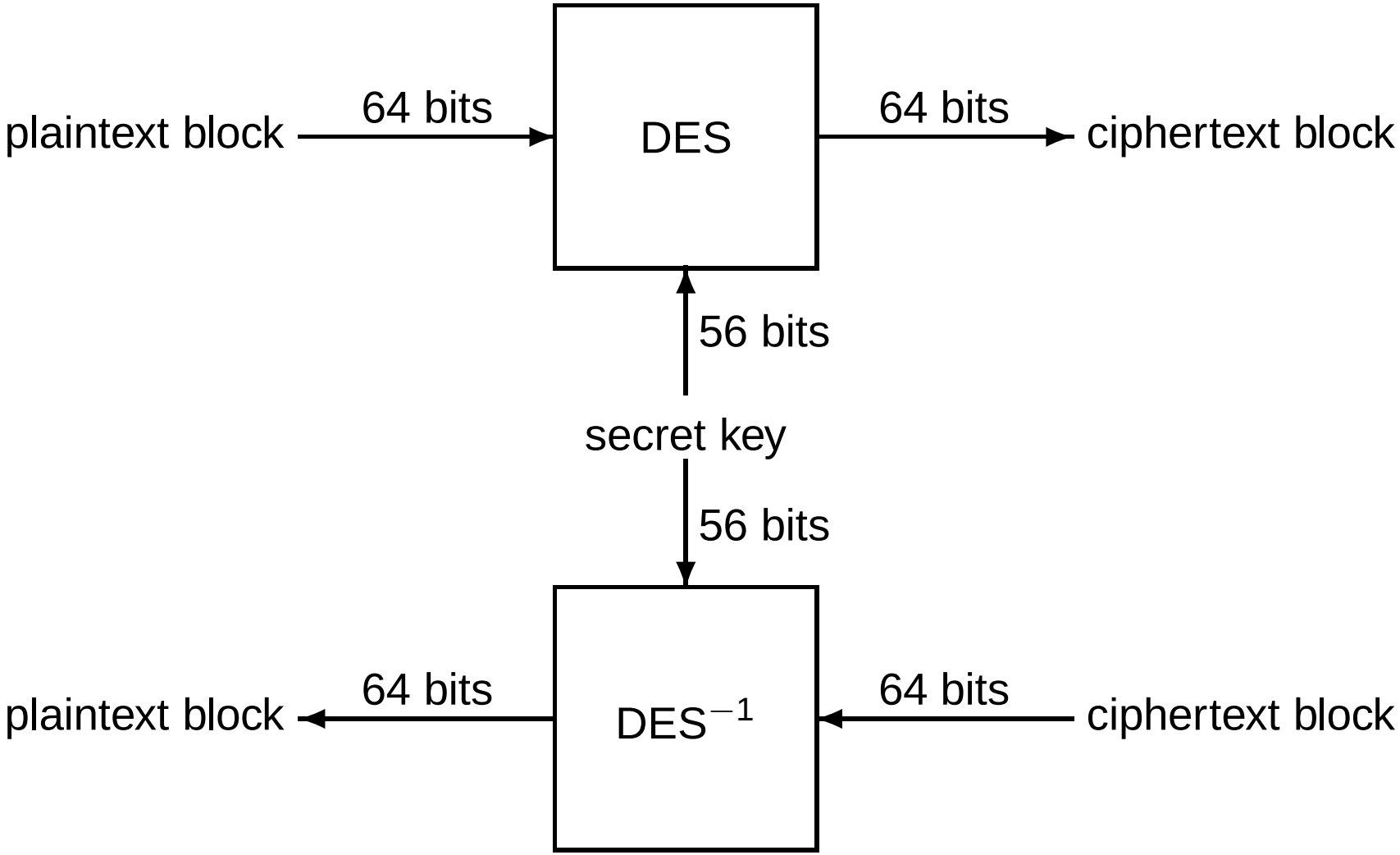
RC4 (Alleged)


Key
key set up
registers i and j
permutation
$S[0], S[1], \ldots, S[255]$
1: $i \leftarrow i + 1$
2: $j \leftarrow j + S[i]$
3: swap $S[i]$ and $S[j]$
4: output $S[S[i] + S[j]]$
output byte

RC4 Key Schedule

1: $j \leftarrow 0$
2: **for** $i = 0$ to 255 **do**
3: $\quad S[i] \leftarrow i$
4: **end for**
5: **for** $i = 0$ to 255 **do**
6: $\quad j \leftarrow j + S[i] + K[i \bmod \ell]$
7: $\quad$ swap $S[i]$ and $S[j]$
8: **end for**
9: $i \leftarrow 0$
10: $j \leftarrow 0$

DES Block Cipher



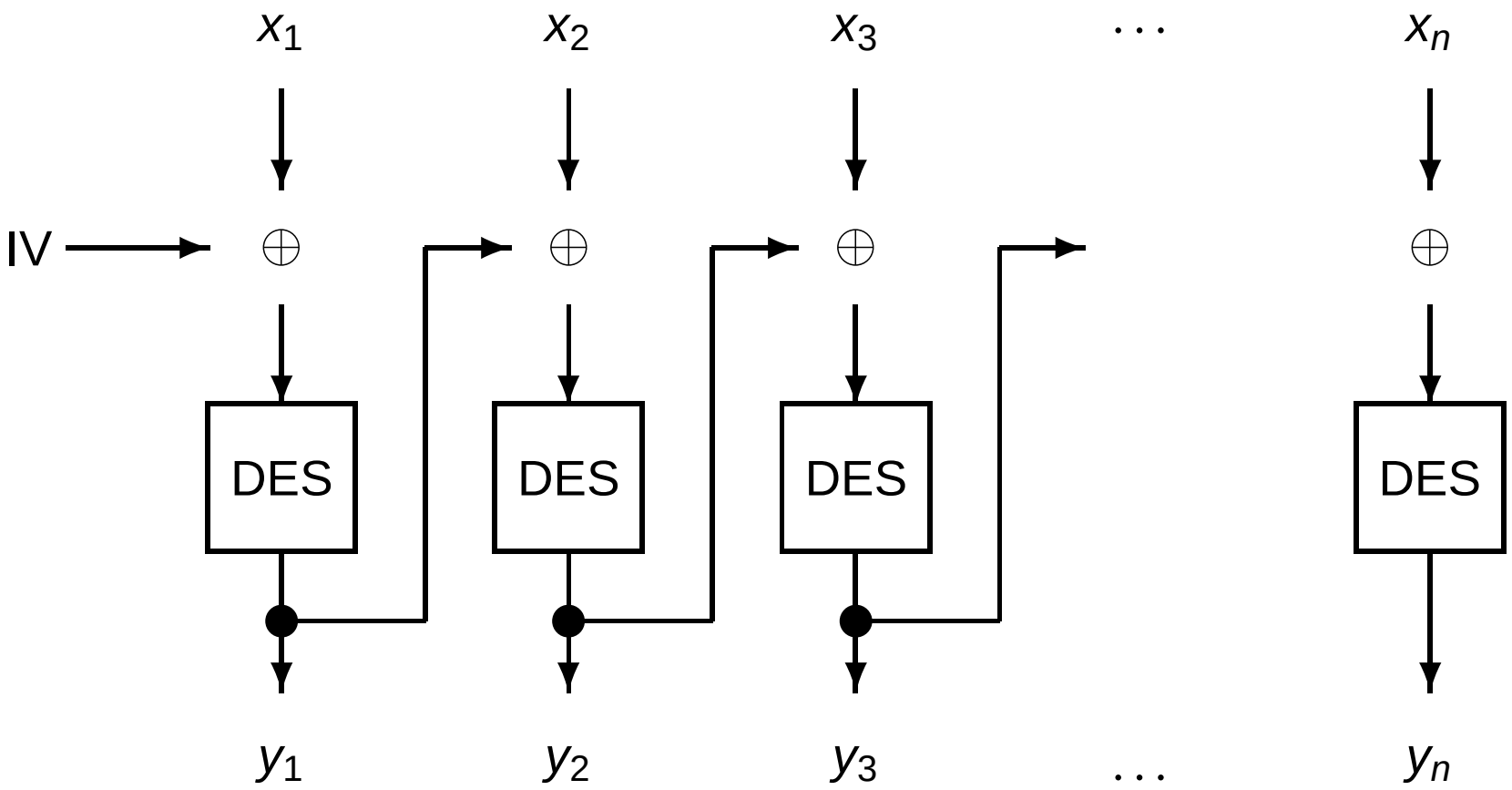
The XOR Operation

$\oplus$	0	1
0	0	1
1	1	0

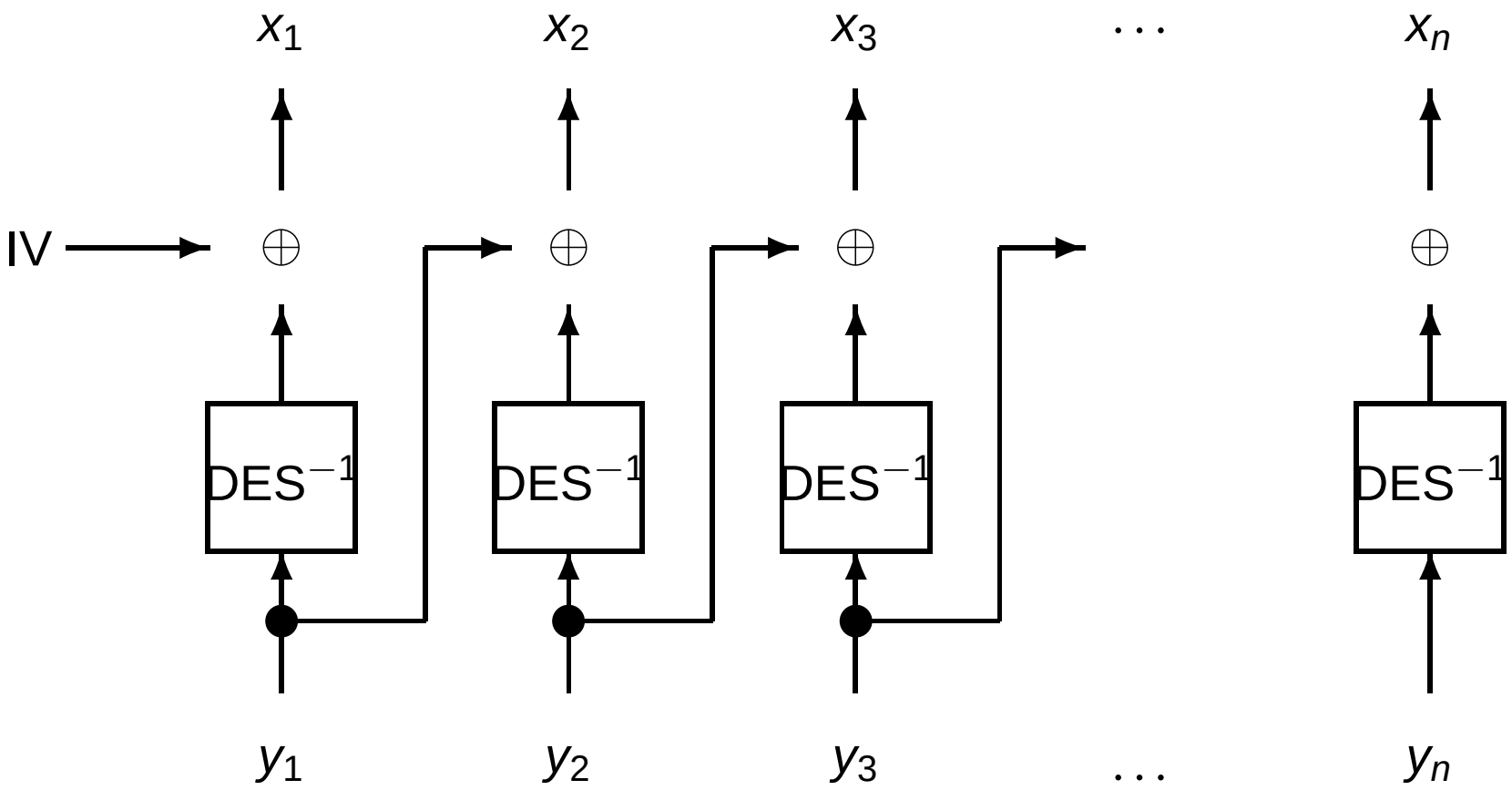
	(X)	10010
$\oplus$	(Y)	00111
$=$	(Z)	10101

$\oplus$	(Y)	00111
$=$	(X)	10010

CBC Encryption Mode



CBC Decryption



Note on the CBC Mode

Three possibilities for dealing with IV

- Using a (non secret) constant IV
- Using a secret IV
- Using a random IV which is sent in clear with the ciphertext

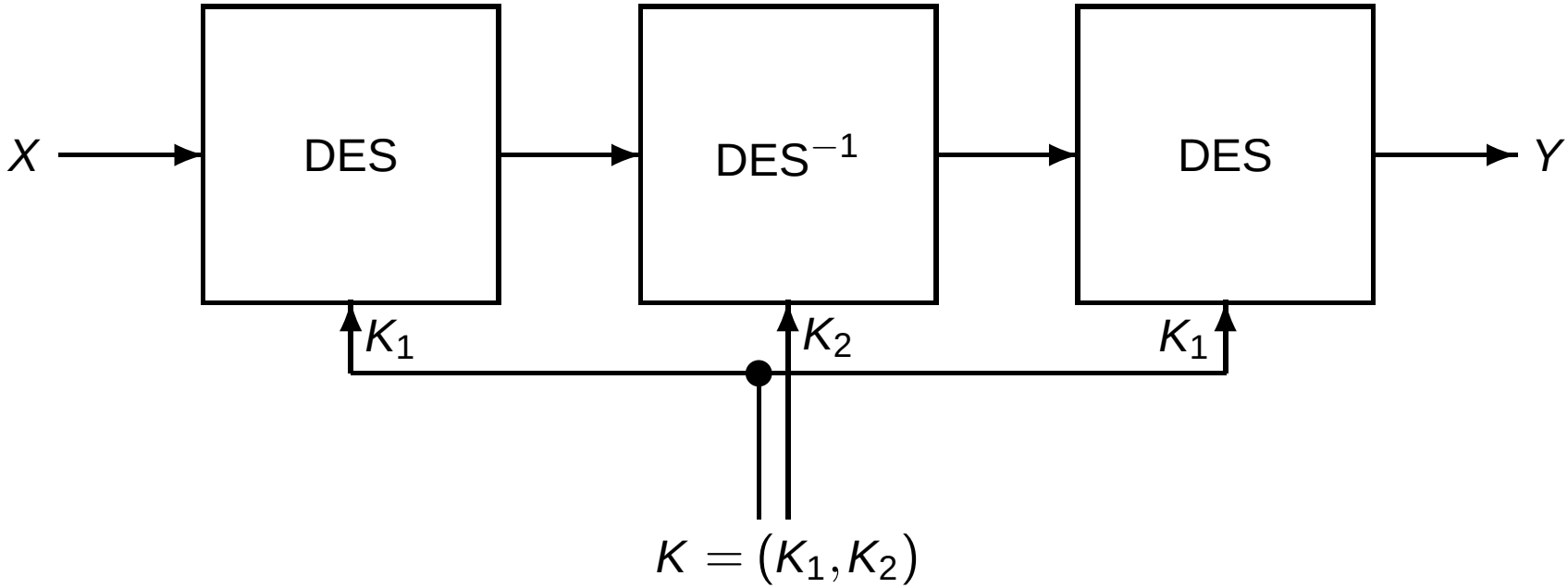
Brute Force Attack on DES

strategy	preprocessing	memory	time
exhaustive search	0	1	2^{56}
dictionary attack	2^{56}	2^{56}	1
tradeoffs	2^{56}	2^{37}	2^{37}

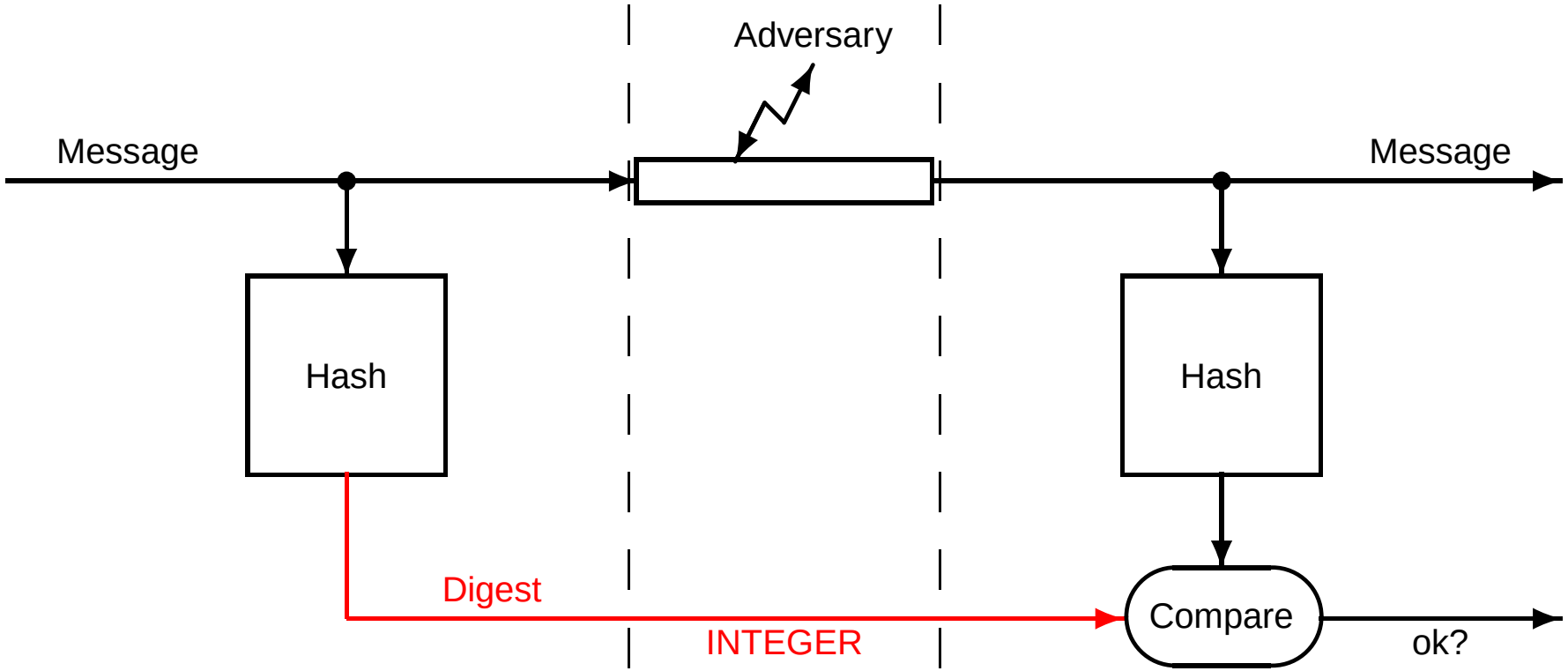
$\longrightarrow$ the key of DES is too short!

$\longrightarrow$ we need some way to enlarge the key

Two-Key Triple DES



Integrity by Hash Function



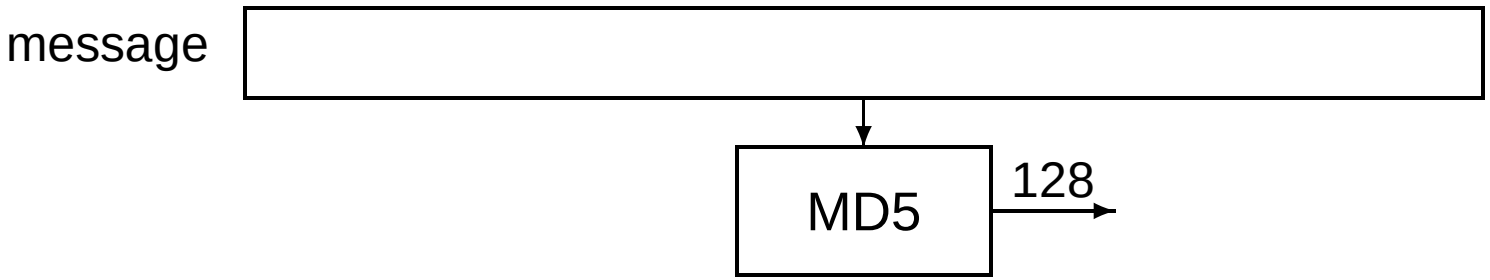
Security of Hash Functions

One-wayness : given y it is hard to find even one x such that
$y = h(x)$.
$\rightarrow$ witness for a password

Collision resistance : it is hard to find x and x' such that
$h(x) = h(x')$ and $x \neq x'$.
$\rightarrow$ digital fingerprint of the bitstring

Randomness : given $h_1(x), \ldots, h_n(x)$ it is hard to predict $h_{n+1}(x)$
$\rightarrow$ secret key generation

Cryptographic Hashing



- “Message Digest” (MD) devised by Ronald Rivest
- “Secure Hash Algorithm” (SHA) standardized by NIST
- MD4 in 1990 (128-bit digest)
- MD5 in 1991 (128-bit digest) published as RFC 1321 in 1992
- SHA in 1993 (160-bit digest) (now obsolete)
- SHA-1 in 1995 (160-bit digest)
- SHA256, SHA384, SHA512 in 2002 (256-, 384-, and 512-bit digest)

Summary of Generic Attacks

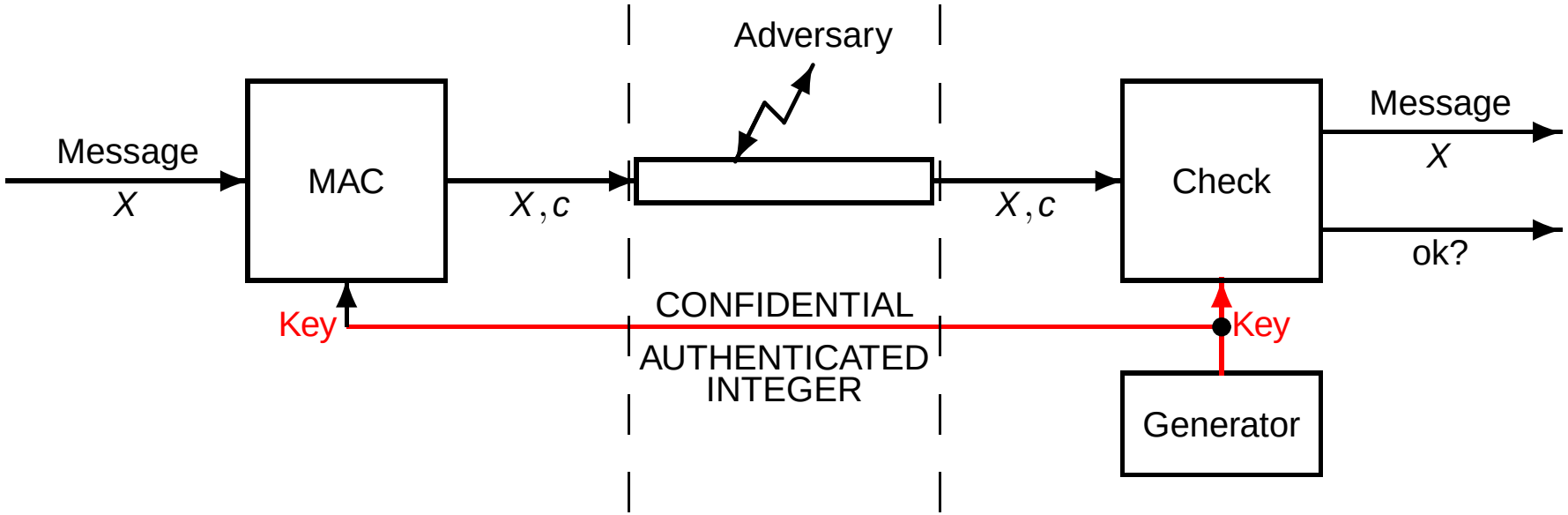
if we hash onto n bits, ($N = 2^n$)

attack	complexity
exhaustive search	2^n
collision attack	$2^{\frac{n}{2}}$

Recent Attacks on Hash Functions

- collision found on MD4 (Dobbertin 1996)
- preimage attack on MD4 (Dobbertin 1997)
- collision found on SHA0 (Joux+ 2004)
- collision found on MD5 (Wang+ 2004)
- theoretical attack on SHA1 (Wang+ 2005)
- ...research going on

Authenticity by Message Authentication Code



Security of Symmetric Encryption

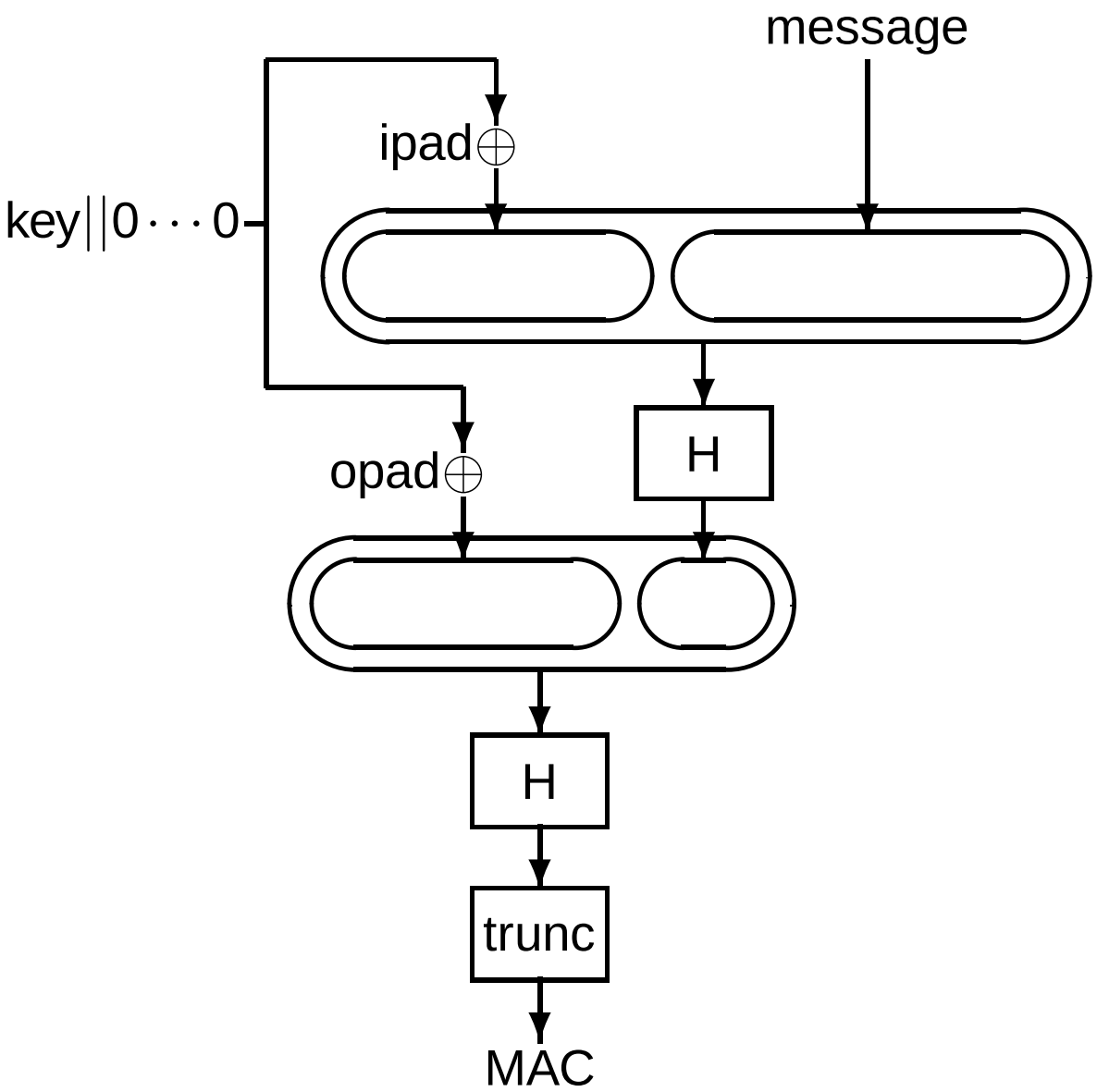
Key-recovery : it is hard to recover the secret key even if we have access to a MAC oracle

Forgery : after playing with a MAC oracle, it is hard to forge a new authenticated message

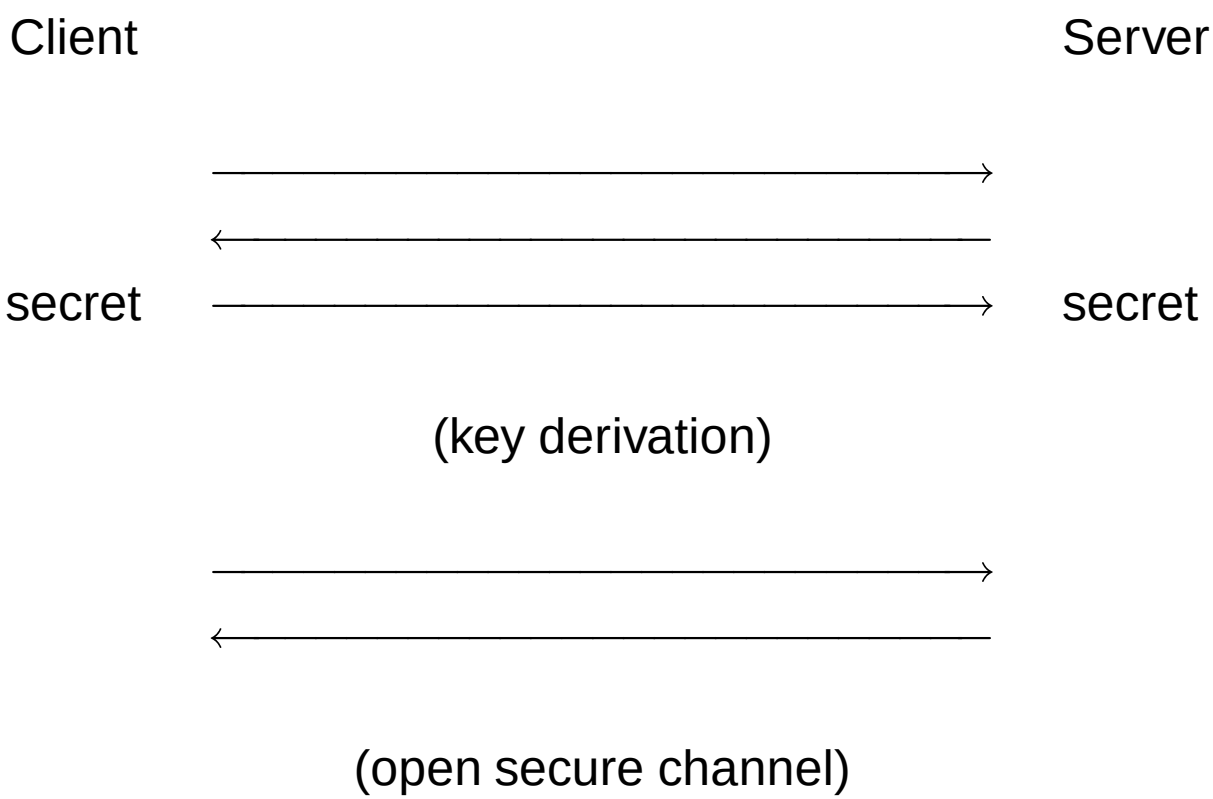
Three Categories of MAC

from stream ciphers	from block ciphers	from hash functions
Wegman-Carter	EMAC	**HMAC**
LFSR-Toeplitz	XCBC	UMAC
bucket hashing	RMAC	...
square hash	TMAC	
...	OMAC	
	...	

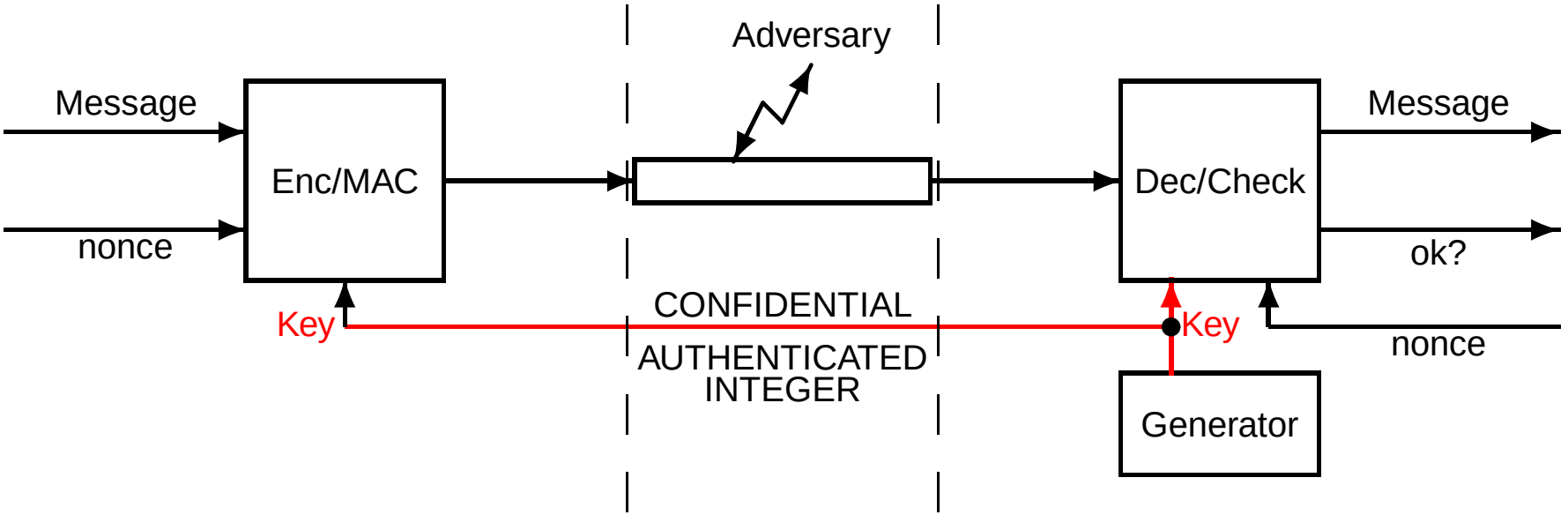
Hashing to Authentication: HMAC [RFC 2104]



A Typical Secure Channel Establishment



Authenticated Modes of Operation



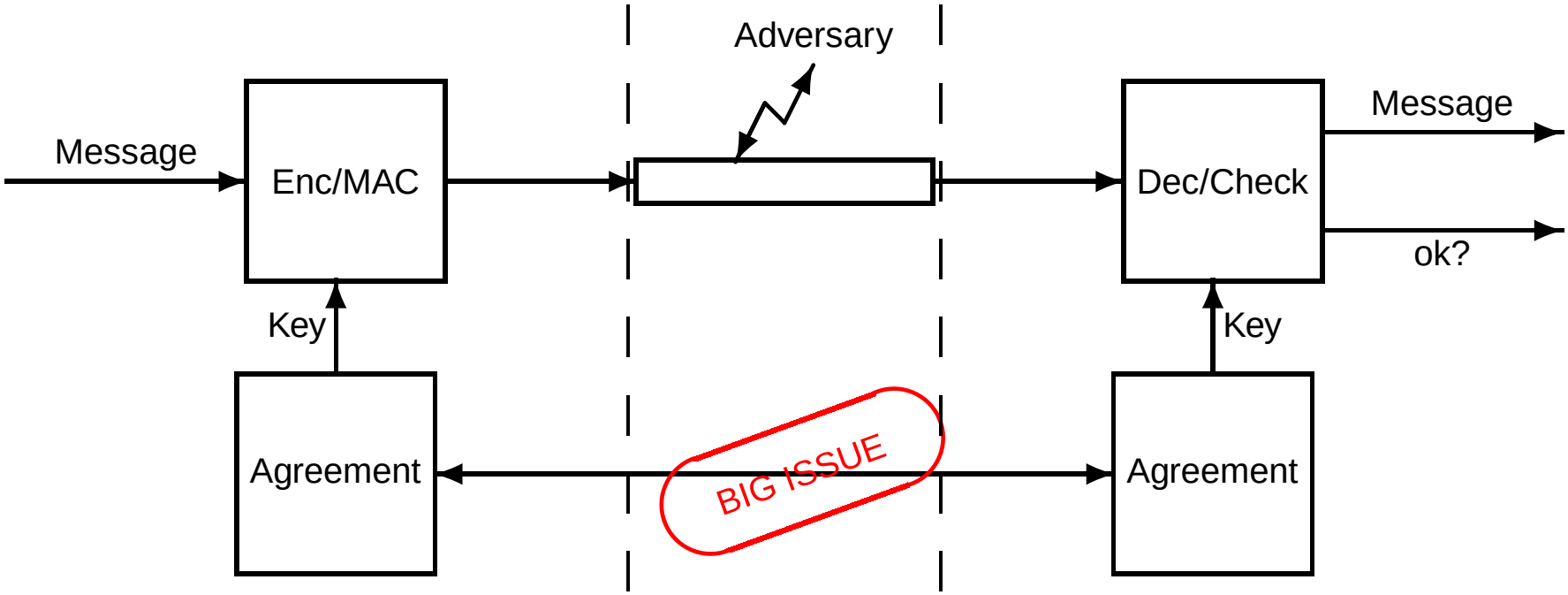
Examples of Secure Channels

- SSL/TLS
- SSH
- IPSEC
- 3GPP
- WPA

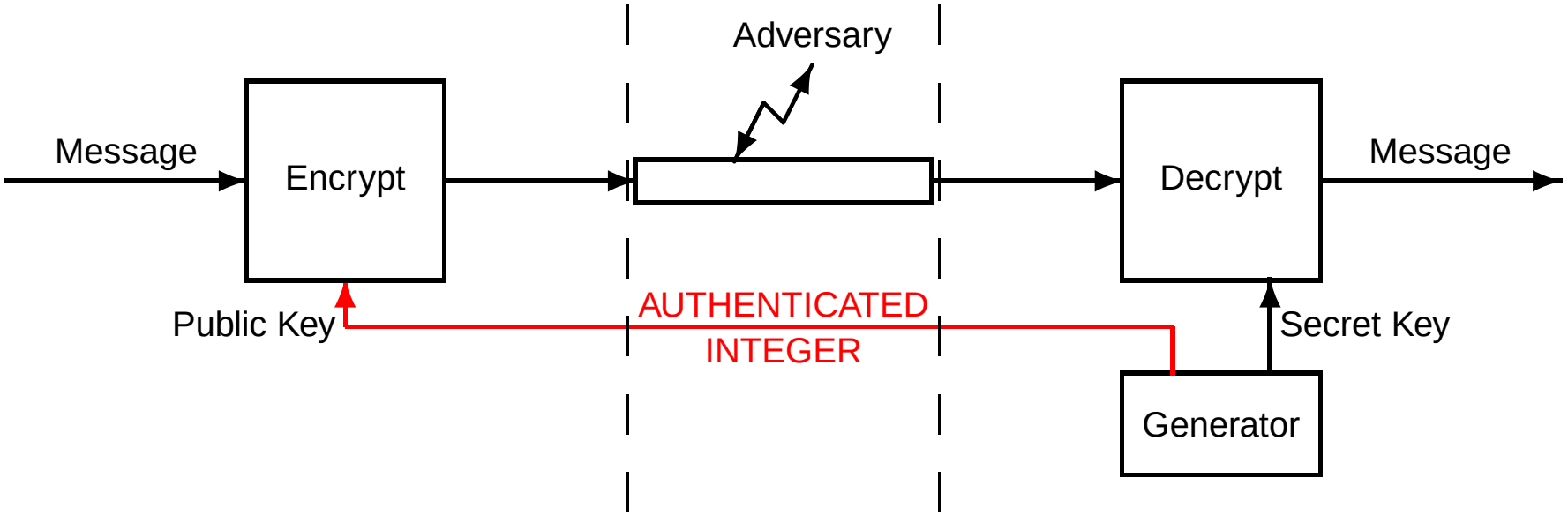
Bad Examples

- GSM
- WEP
- Bluetooth

Remaining Problem: Key Setup



Public-Key Cryptosystem

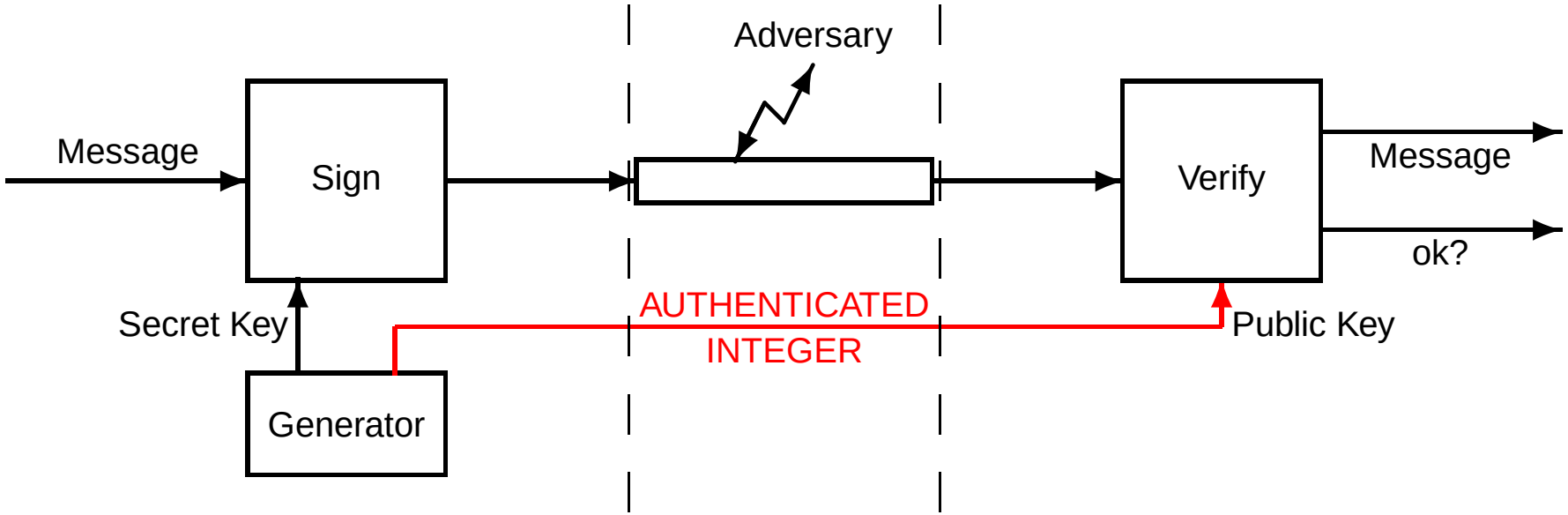


Security

Semantic security: given a public key, for any two plaintexts, we cannot distinguish a valid encryption of either plaintext (apply to probabilistic encryption only)

Message recovery: given a public key and the encryption of a random plaintext, it is hard to recover the plaintext (consequence of semantic security)

Digital Signature

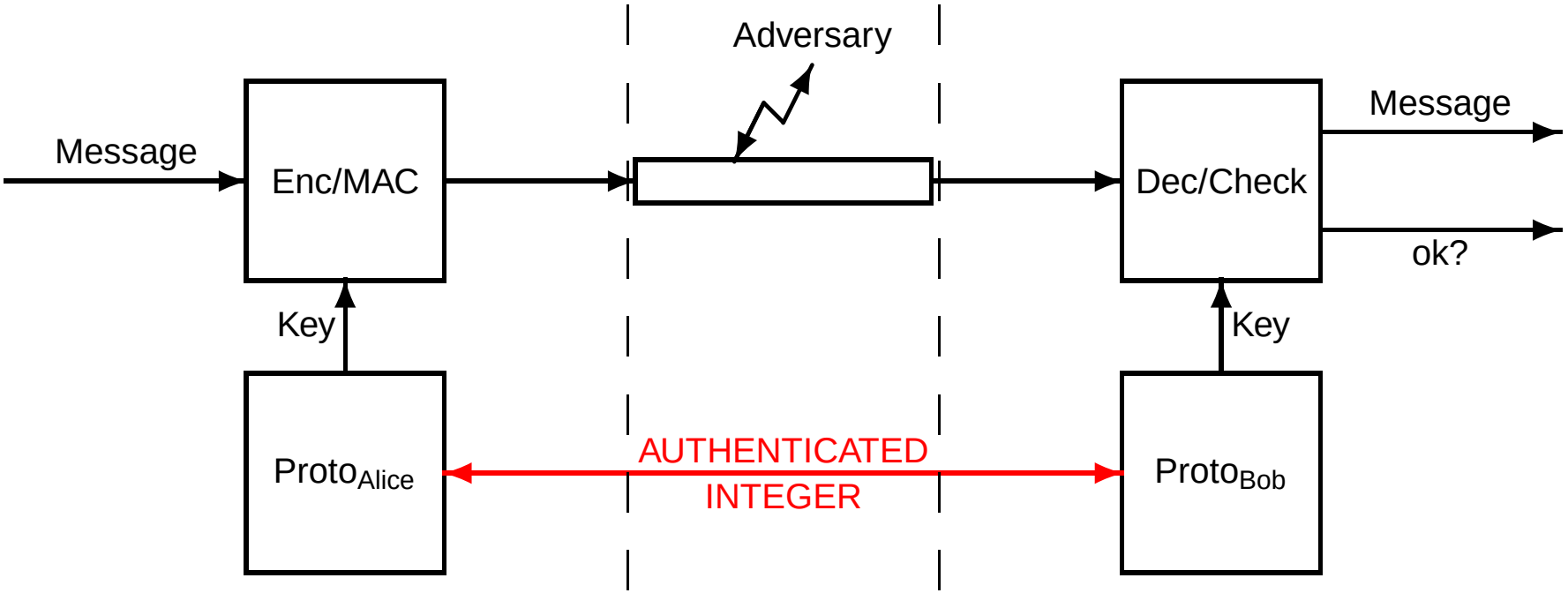


Security

Unforgeability: given a public key, it is hard to forge a new valid message-signature pair

Non-repudiation: given a public key, any valid message-signature pair must have been created by the secret key holder (often based on unforgeability)

Key Exchange Protocol



Security

Secrecy: by looking at the communication protocol, it is impossible to guess the exchanged key

The Diffie-Hellman Key Agreement Protocol

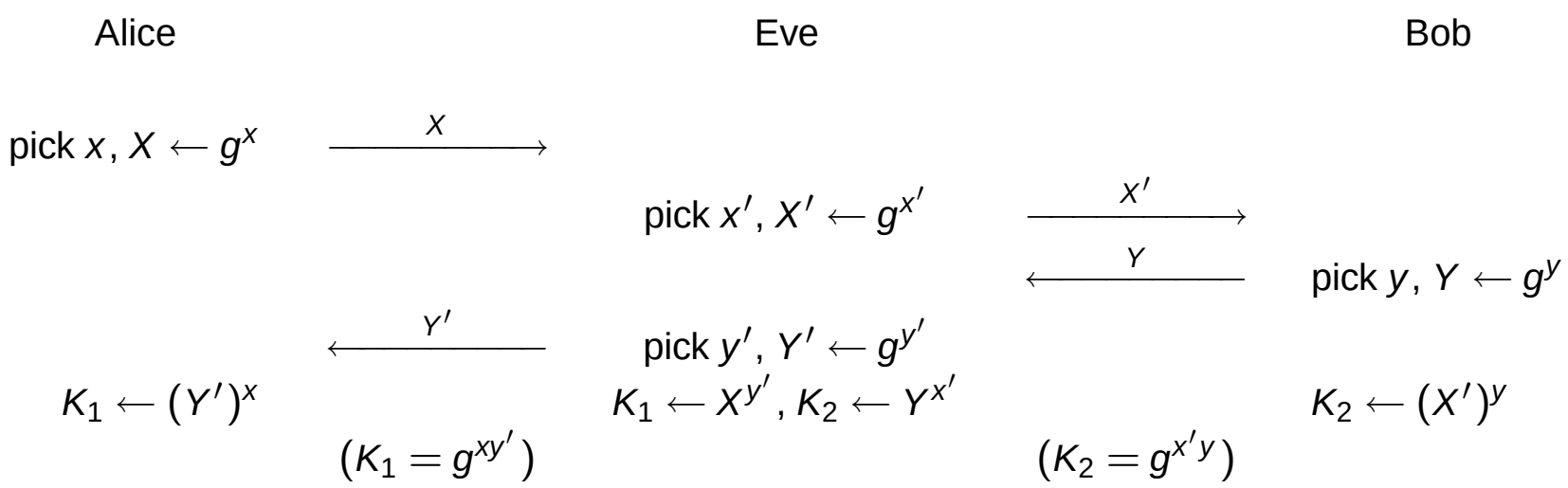
Assume a group ($\mathbf{Z}_p^*$, elliptic curves, ...) generated by some g

Alice		Bob
pick x at random		
$X \leftarrow g^x$	$\xrightarrow{X}$	
	$\xleftarrow{Y}$	pick y at random
		$Y \leftarrow g^y$
$K \leftarrow Y^x$		$K \leftarrow X^y$

$$(K = g^{xy})$$

communications must authenticated and integer!

If we Lack Authentication: Man-in-the-Middle Attack



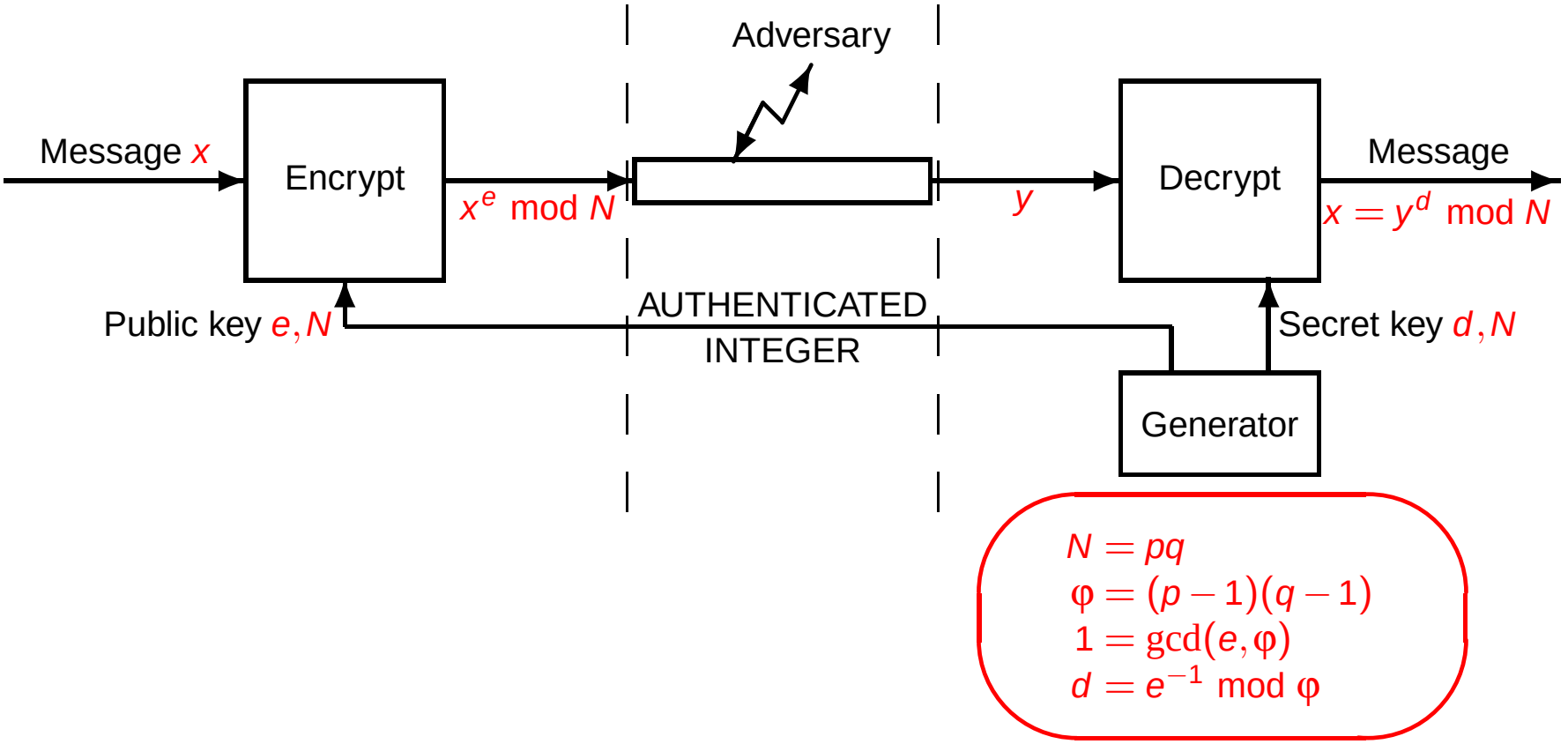
Static versus Ephemeral Diffie-Hellman

- Ephemeral DH: it provides **forward secrecy**

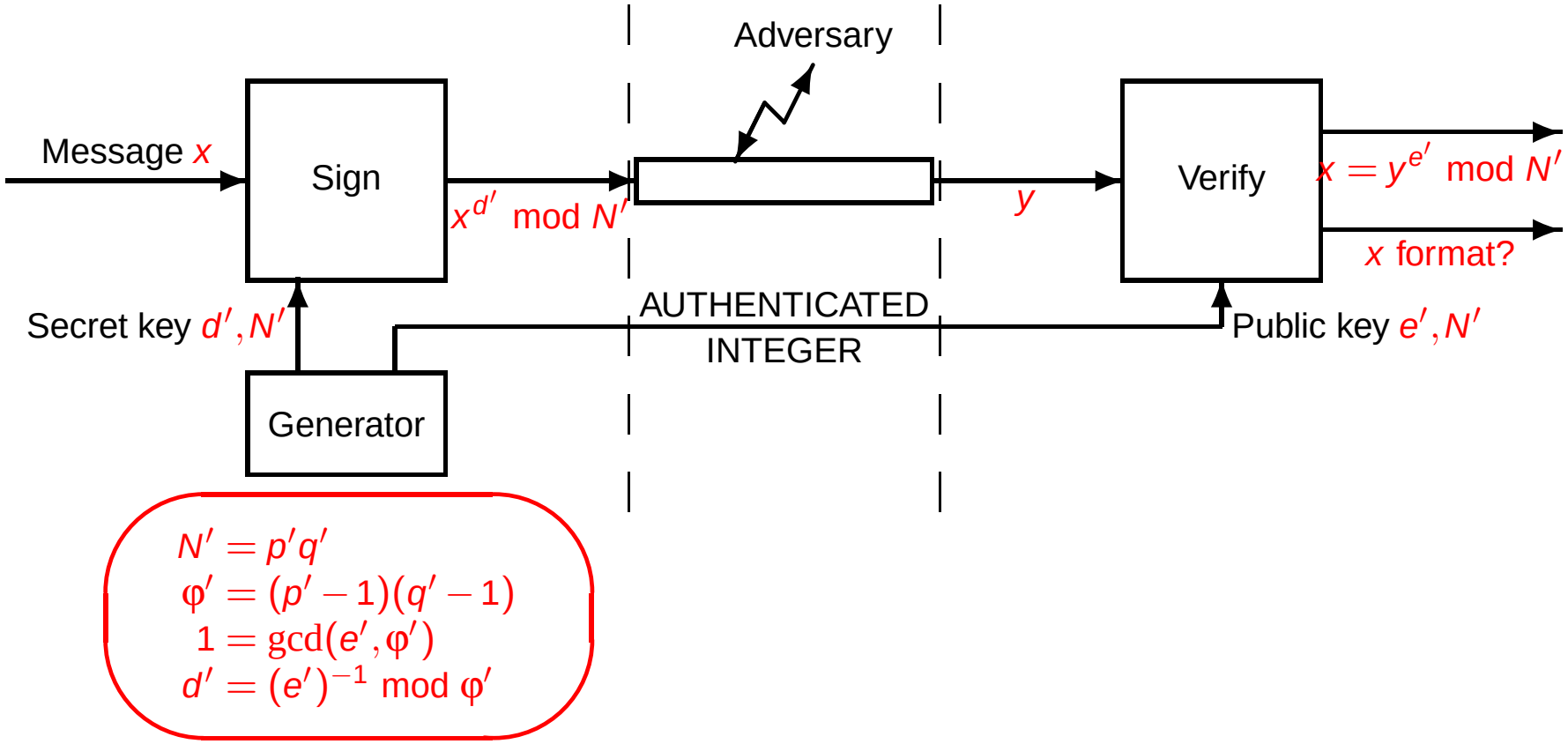
  "if long-term secret keys are compromised at time t, this does not compromise a DH session key at time $t' < t$"

- Static DH: X and Y are used like public keys

Plain RSA Encryption



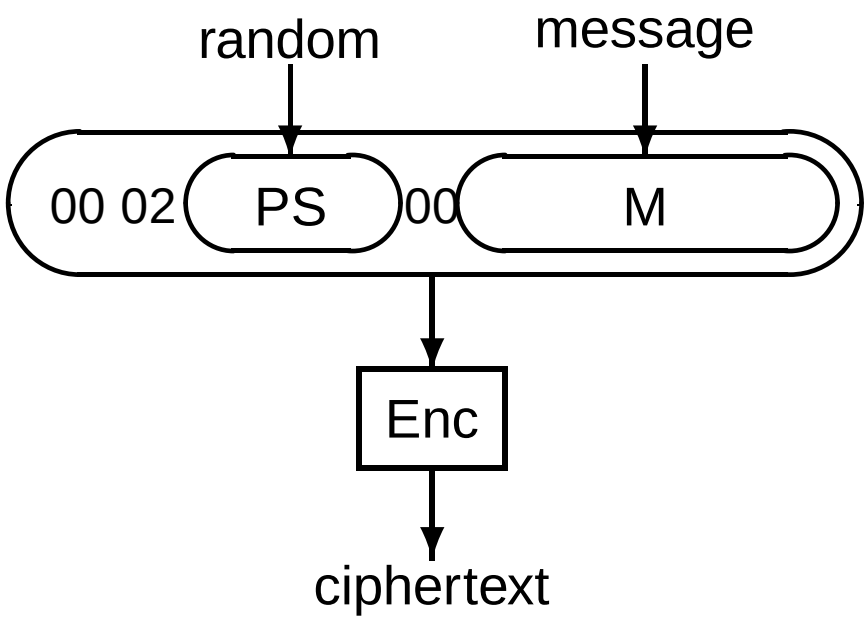
Plain RSA Signature



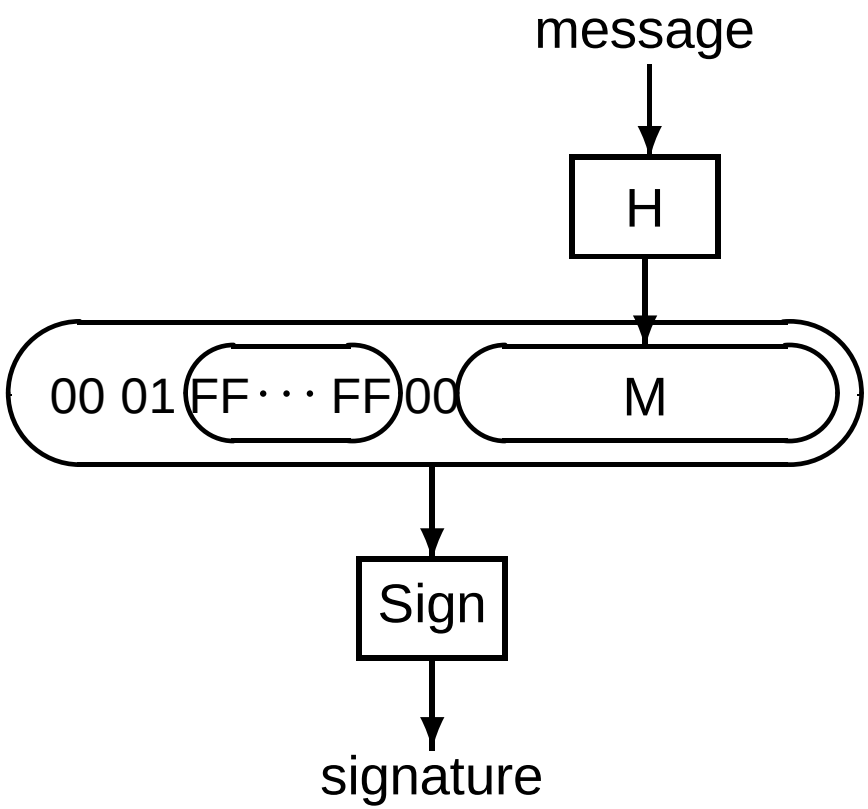
Special Application Attacks

- short exponents
- messages with known structure
- related messages
- related keys
- ...

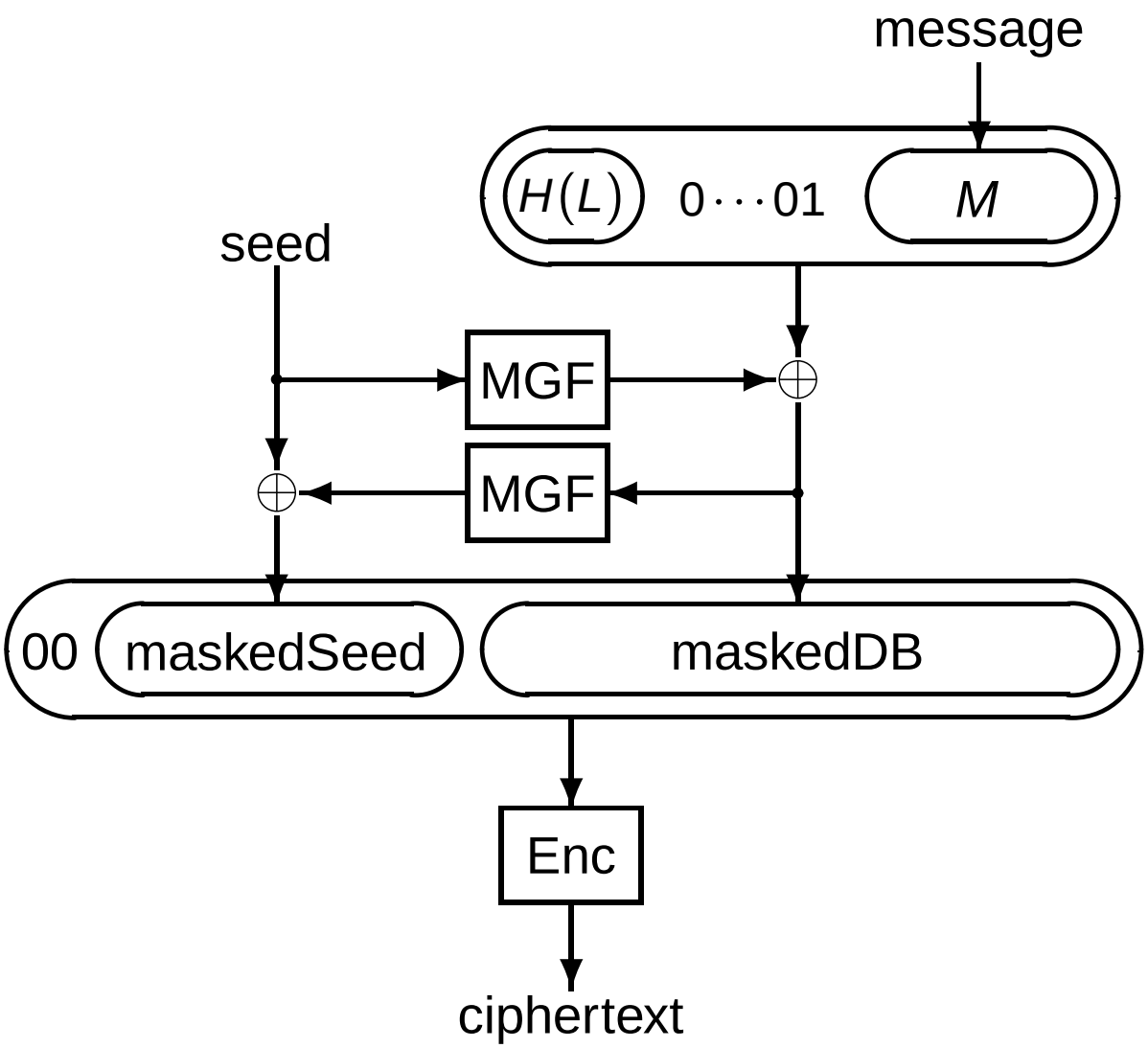
PKCS#1v1.5 Encryption



PKCS#1v1.5 Signature



RSA-OAEP


message
H(L)
0···01
M
seed
MGF
MGF
00
maskedSeed
maskedDB
Enc
ciphertext

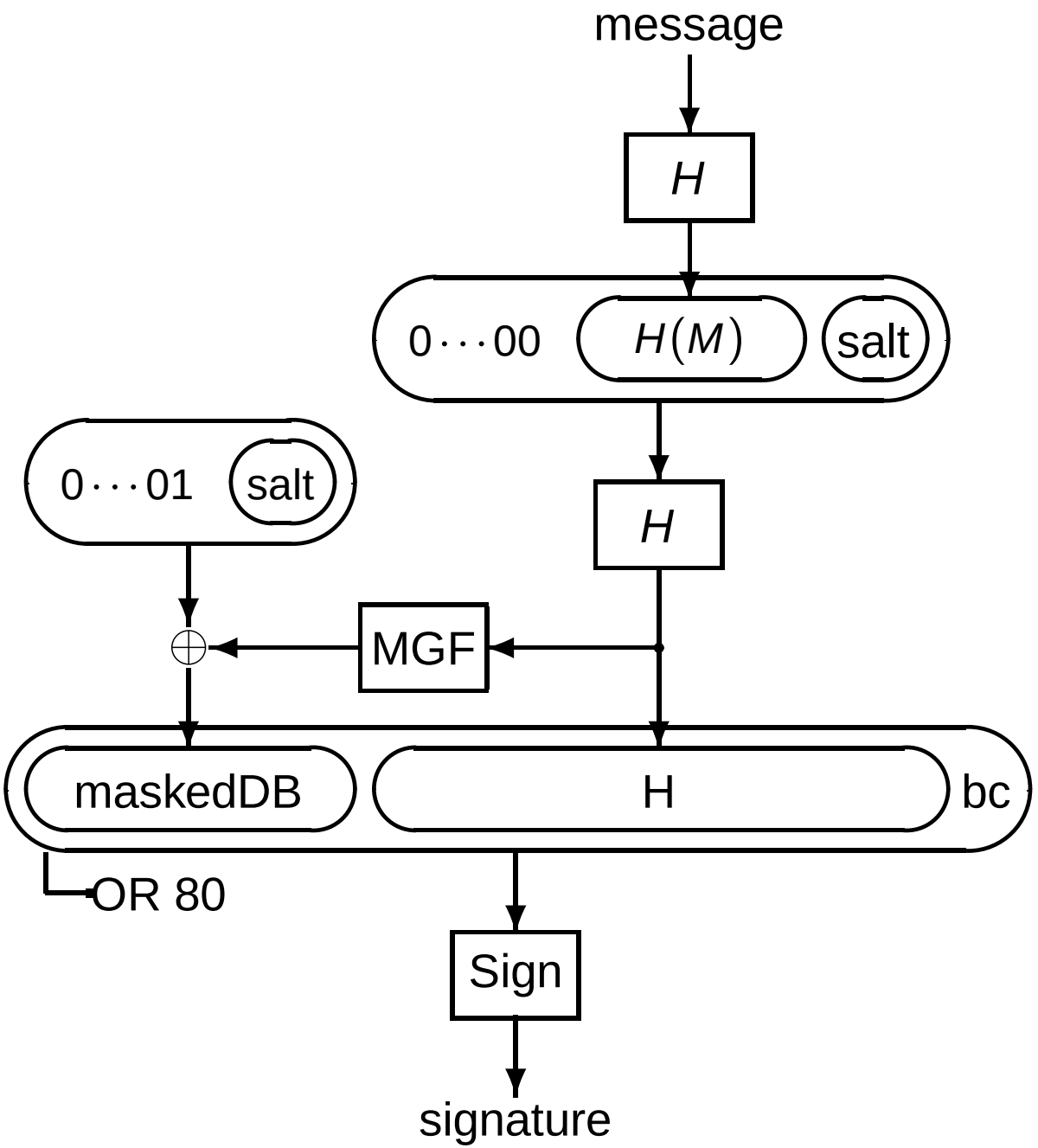
Mask Generation Function in RSA-OAEP

The PKCS specifications further suggests an mask generation function MGF1 which is based on a hash function. The $\mathrm{MGF1}_\ell(x)$ string simply consists of the ℓ leading bytes of

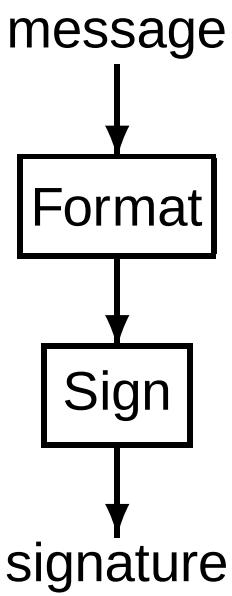
$$H(x||00000000)||H(x||00000001)||H(x||00000002)||\cdots$$

in which x is concatenated to a four-byte counter.

RSA-PSS

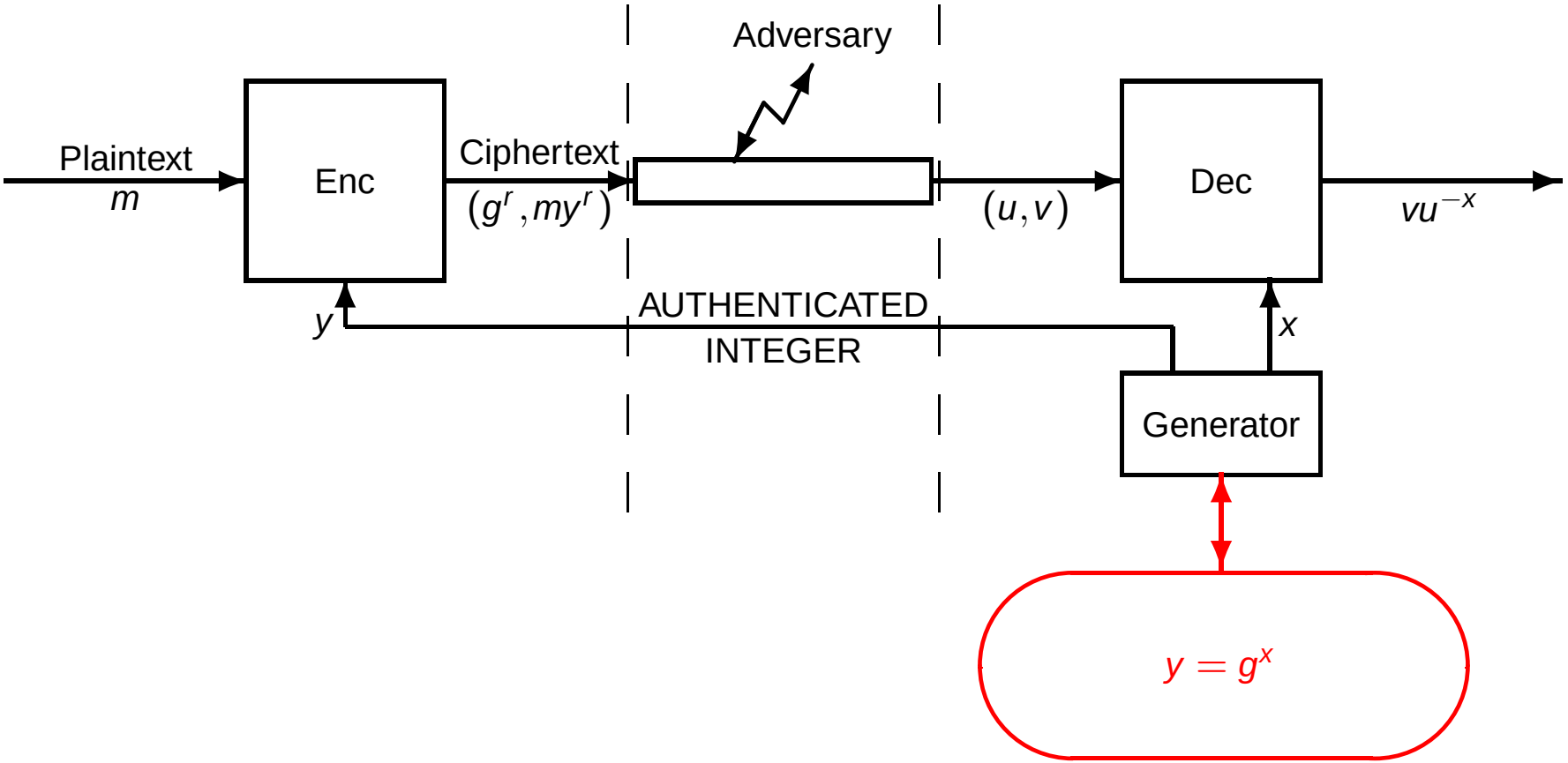

message
H
0···00
H(M)
salt
0···01
salt
H
MGF
maskedDB
H
bc
OR 80
Sign
signature

ISO/IEC 9796 Signature

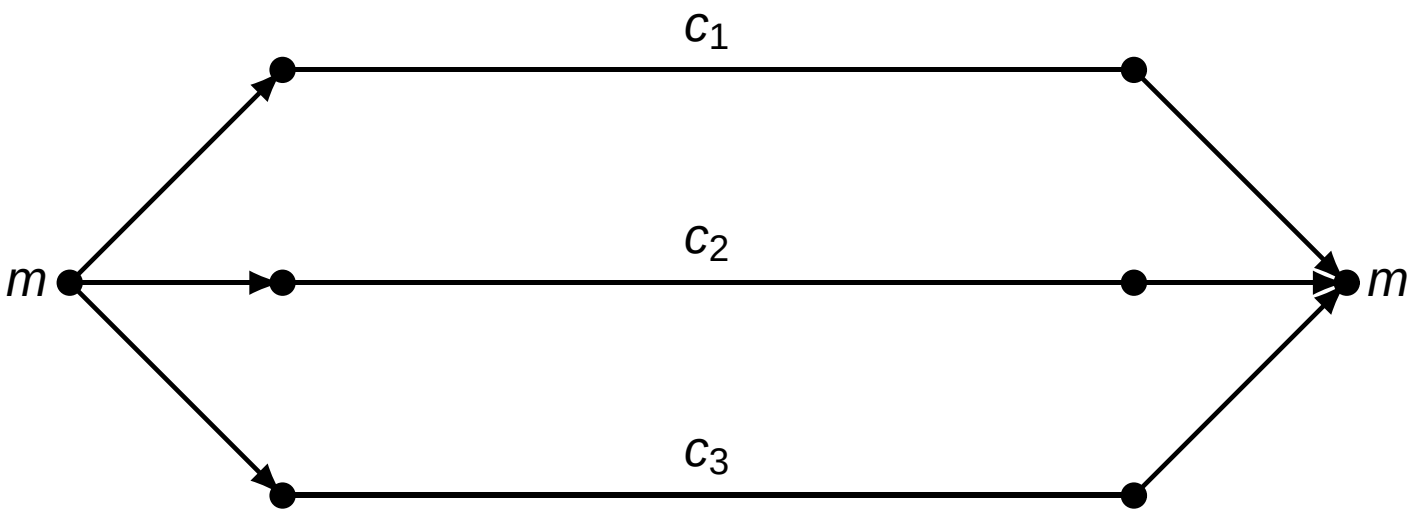


- format is invertible
- signature with message recovery

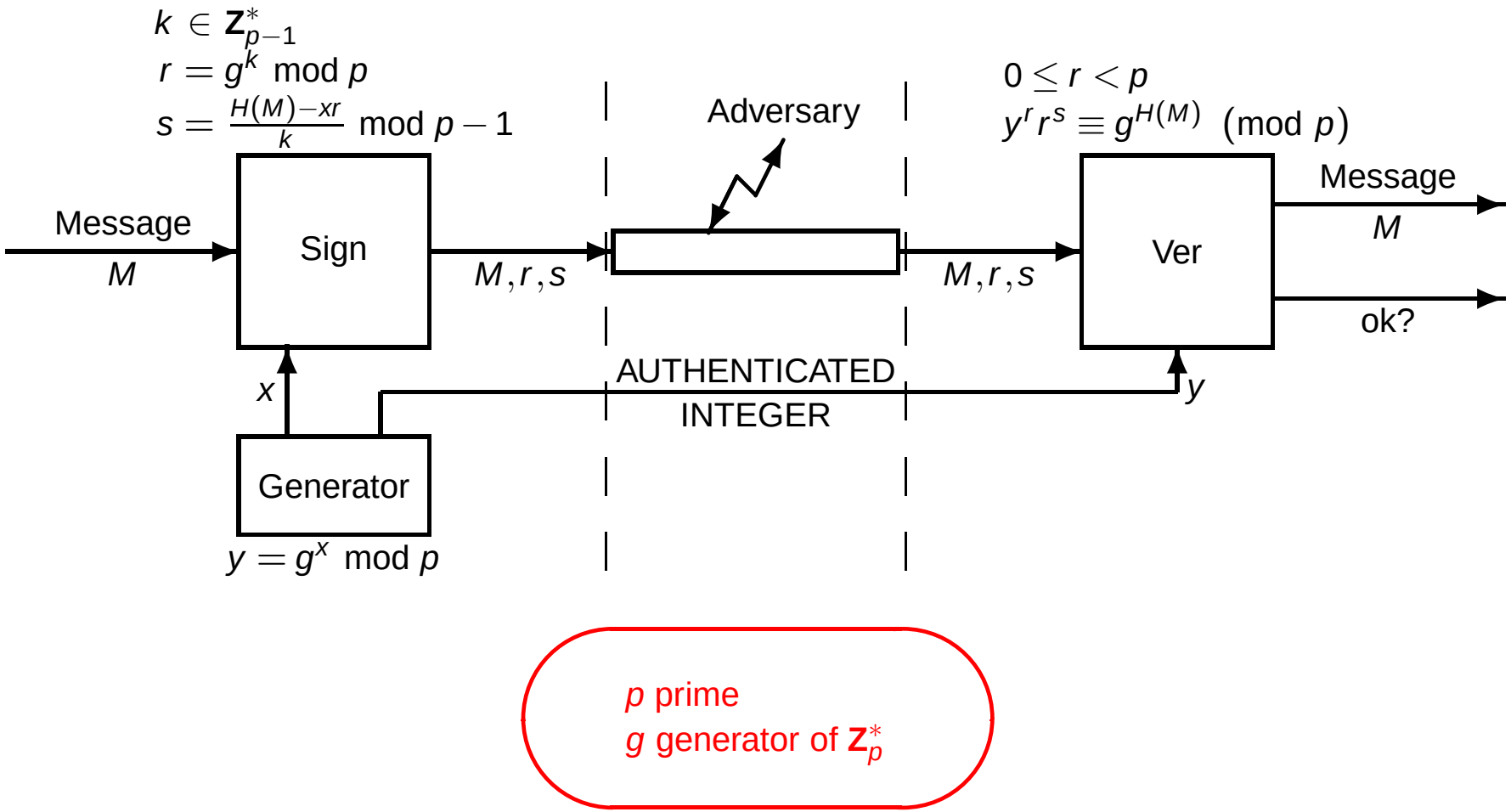
Plain ElGamal Encryption



Non-Deterministic Encryption



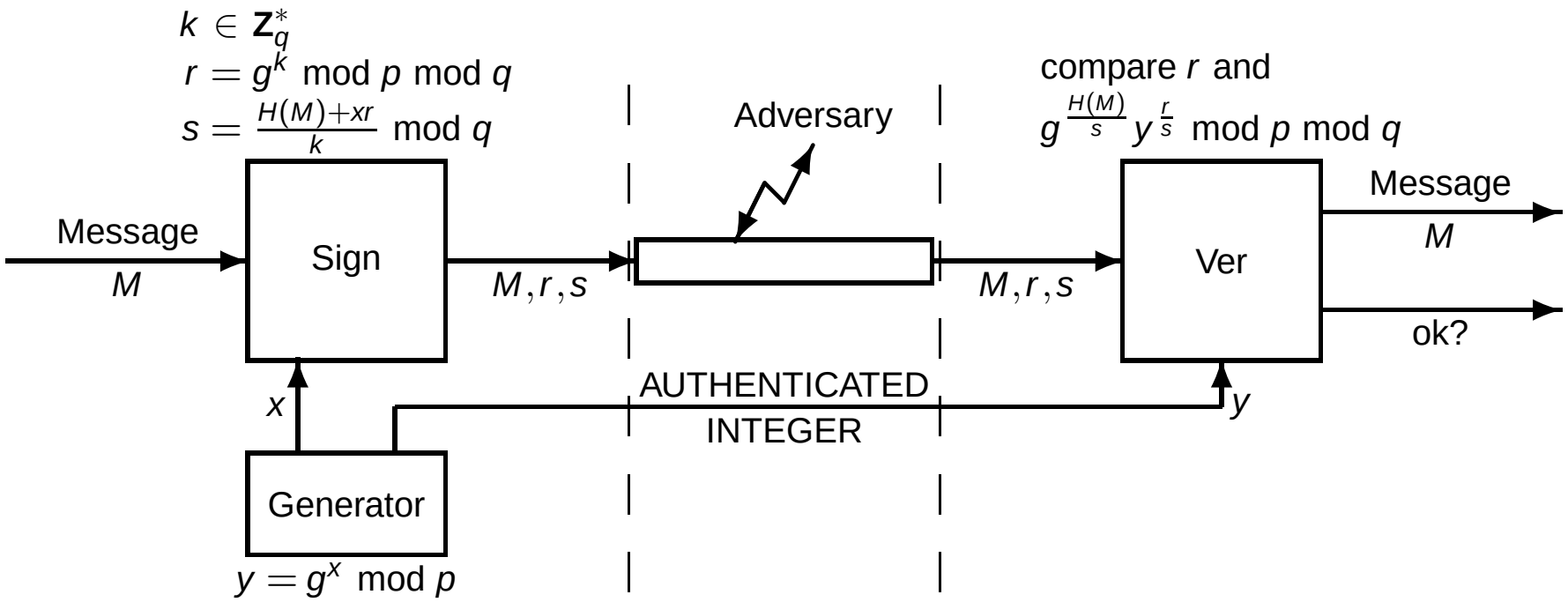
ElGamal Signature



The ElGamal Dynasty

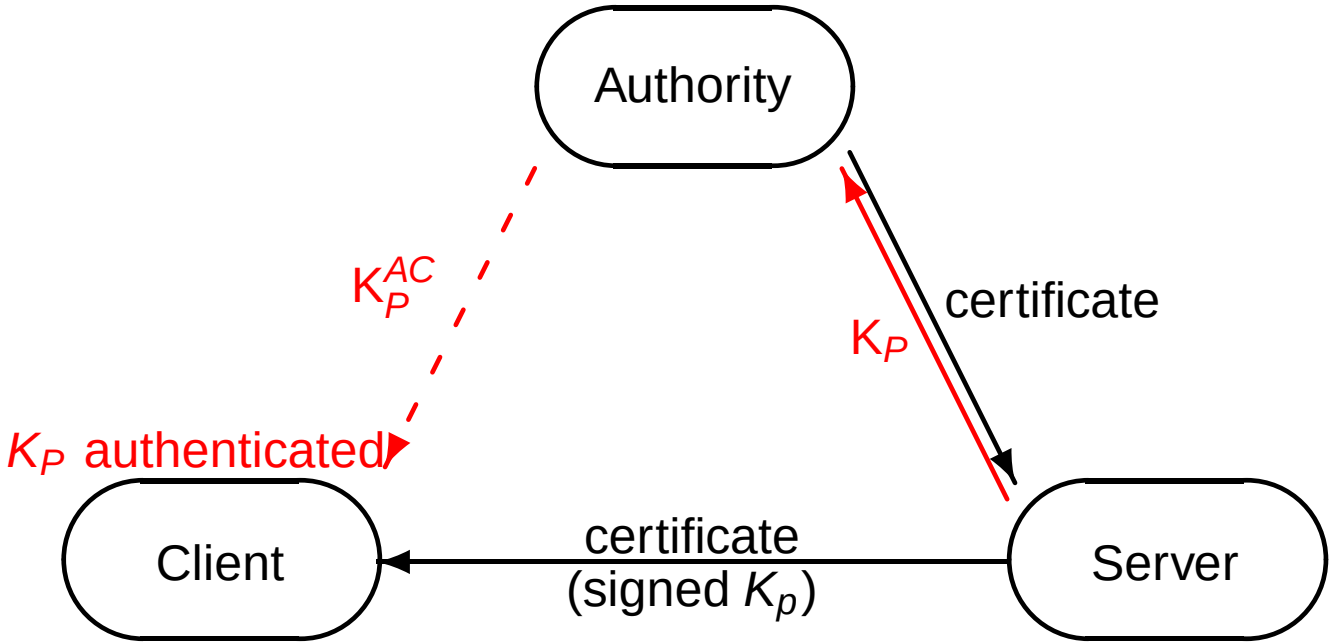
- 1984 ElGamal signatures
- 1989 Schnorr signatures
- 1995 DSA: US signatures
- 1995 Nyberg-Rueppel signatures
- 1997 Pointcheval-Vaudenay signatures
- 1998 KCDSA: Korean signatures
- 1998 ECDSA
- ...

DSA Signature



q prime
$p = aq + 1$ prime
$g = \text{random}^a \bmod p > 1$

Public-Key Infrastructure



An X.509 Certificate Example: Overall Structure

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 674866 (0xa4c32)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ZA, ST=Western Cape, L=Cape Town,
            O=Thawte Consulting cc, OU=Certification Services Division,
            CN=Thawte Server CA/Email=server-certs@thawte.com
        Validity
            Not Before: Jun  2 13:10:11 2003 GMT
            Not After : Jun 11 10:21:15 2005 GMT
        ...
        X509v3 extensions:
            X509v3 Extended Key Usage: TLS Web Server Authentication
            X509v3 Basic Constraints: critical CA:FALSE
    Signature Algorithm: md5WithRSAEncryption
        8d:7b:78:60:88:c4:13:4e:94:0d:bc:3b:1b:1c:b6:c9:bc:b1:
        0b:ed:7d:eb:6f:08:3a:ba:6d:21:36:93:38:36:66:7b:a7:bc:
        c0:3f:c4:e0:cf:b4:02:58:be:a6:b9:1d:45:a2:c4:58:38:07:
        e4:63:1a:d9:b9:8d:27:7c:93:67:31:82:6f:a3:3c:86:0c:e0:
        10:71:de:f2:e9:74:af:ac:76:b4:5b:8e:48:57:9d:8f:12:f6:
        72:63:8a:79:b4:74:e0:ba:ca:ac:1a:36:b4:16:38:c1:c5:d2:
        73:ed:e8:64:b0:ae:9e:e2:36:d7:0c:77:92:cc:c7:c0:e0:8a:
        54:24
```

An X.509 Certificate Example: Subject

```
Subject: C=CH, ST=Bern, L=Bern,
    O=Switch - Teleinformatikdienste fuer Lehre und Forschung,
    CN=nic.switch.ch
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
        Modulus (1024 bit):
            00:d0:0e:b7:16:bf:86:59:c3:97:e6:02:33:59:90:
            65:29:b0:69:73:64:83:03:1b:df:62:a8:4d:c0:4f:
            3c:d9:12:6b:8c:57:95:e1:57:e8:48:a6:7f:dd:15:
            8b:9d:ad:93:dc:78:af:06:1a:ce:0f:7b:cc:c4:6f:
            a0:06:26:40:73:04:d3:da:7b:20:c1:15:37:8c:2f:
            58:c4:d4:c1:4b:18:84:5c:54:f1:b1:a0:44:3c:e2:
            0e:8a:a2:63:48:6b:34:c7:10:9d:a1:23:56:77:f5:
            4e:3d:38:9a:70:5e:03:02:30:45:ee:81:e4:94:96:
            47:18:9e:47:37:bb:18:f6:87
        Exponent: 65537 (0x10001)
```

History

- First version by Netscape in 1994
- Microsoft version PCT in 1995
- SSLv3 by Netscape in 1995
- IETF version TLS/1.0 in 1997 [RFC2246]
- IETF version TLS/1.1 in 2005 (draft) [RFC2246]

Goal: secure any communication (e.g. HTTP) based on TCP/IP

Common Use Principle

- client-server communications, random client, corporate server
- trusted third party: certificate authority (CA)
- A+I secure channel with CA to be used only once
- authentication of server based on public key
- authentication of client (if needed) based on password
- interoperable cipher suites

TLS Record Protocols

- Handshake Protocol (for initiating a session)
- Change Cipher Spec Protocol (for setting up cryptographic algorithms)
- Alert Protocol (for managing warnings and fatal errors)
- Application Data Protocol

Session State

- Session identifier
- Peer certificate (if any)
- Cipher suite choice
  - Algorithm for authentication and key exchange during handshake
  - Cipher Spec: symmetric algorithms (encryption and MAC)
- Master secret (a 48-byte symmetric key)
- nonces (from the client and the server)
- sequence numbers (one for each communication direction)
- compression algorithm (if any)

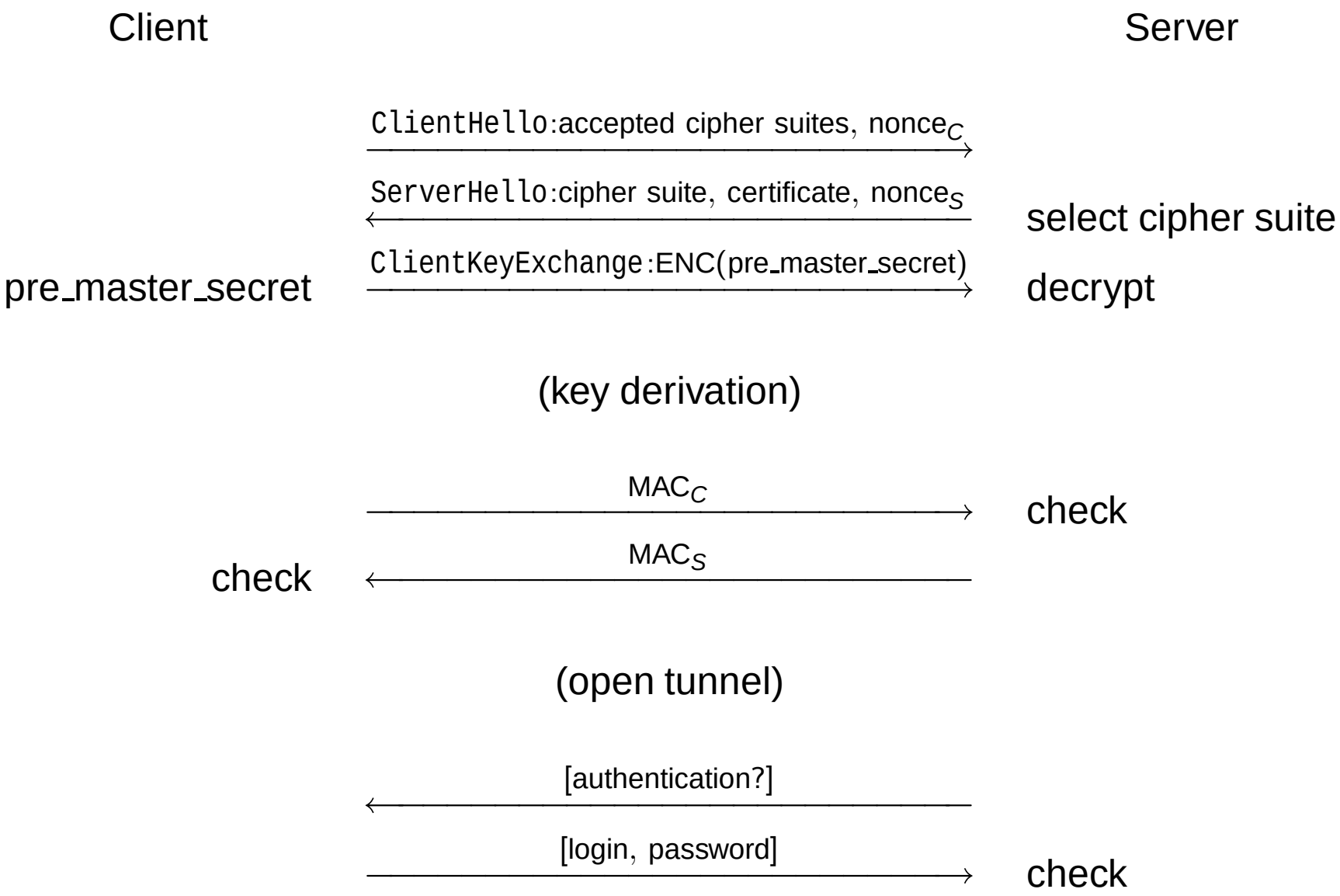
Original TLS Cipher Suites — i

CipherSuite	Key Exchange	Cipher	Hash
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA-1
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4_40	MD5
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA-1
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA	RC2_40	MD5
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA	SHA-1
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DES40	SHA-1
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES	SHA-1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE	SHA-1
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	DH_DSS	DES40	SHA-1
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES	SHA-1
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE	SHA-1
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	DH_RSA	DES40	SHA-1
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES	SHA-1
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE	SHA-1

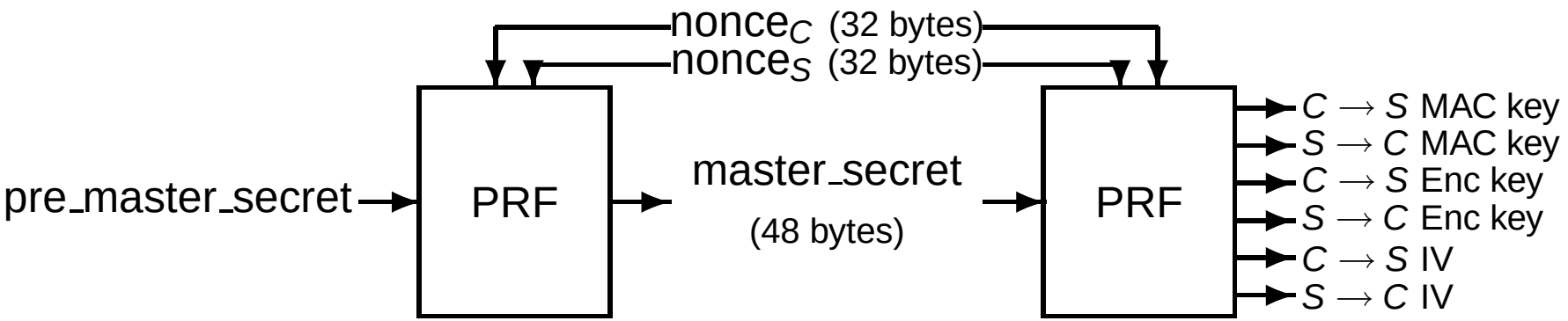
Original TLS Cipher Suites — ii

CipherSuite	Key Exchange	Cipher	Hash
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DHE_DSS	DES40	SHA-1
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES	SHA-1
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE	SHA-1
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	DHE_RSA	DES40	SHA-1
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA-1
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE	SHA-1
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	DH_anon	RC4_40	MD5
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon	DES40	SHA-1
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES	SHA-1
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE	SHA-1

A Typical TLS Session



Key Derivation in SSL/TLS



PRF

Given a secret, a seed, and a string label we define a sequence

$$
\begin{aligned}
a_0 &= \text{seed} \\
a_i &= \text{HMAC}_{\text{hash}}(S, a_{i-1}) \\
r_i &= \text{HMAC}_{\text{hash}}(S, a_i || \text{seed}) \\
\text{P\_hash}(S, \text{seed}) &= r_1, r_2, r_3, \ldots \\
\text{PRF}(\text{secret}, \text{label}, \text{seed}) &= \text{P\_MD5}(S1, \text{label} || \text{seed}) \oplus \\
&\quad\ \text{P\_SHA1}(S2, \text{label} || \text{seed})
\end{aligned}
$$

where $S1$ and $S2$ are the two halves of secret.
(If secret has an odd length, its middle byte is both the last byte of $S1$ and the first byte of $S2$.)

Using PRF

We define

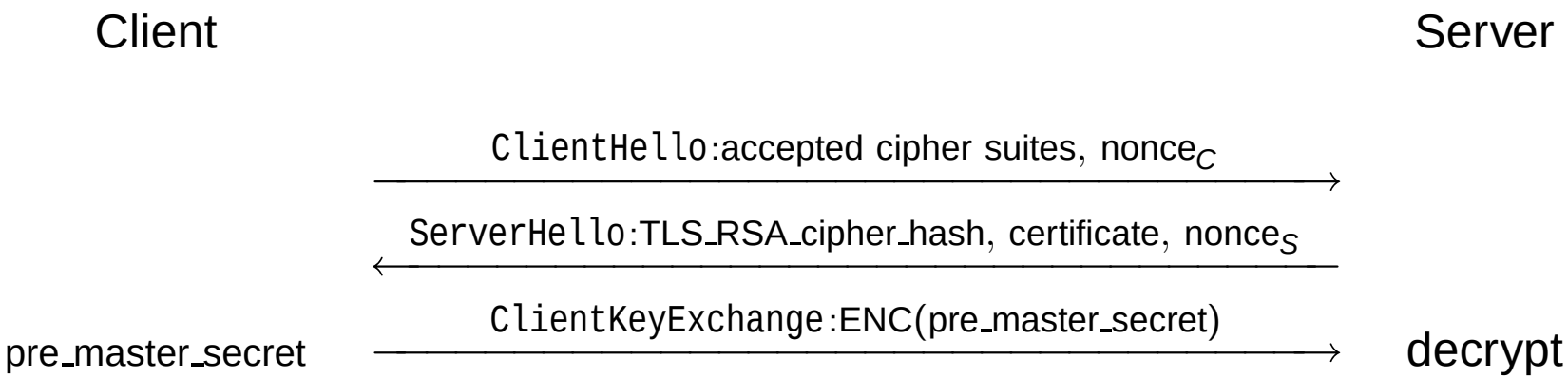
$$
\begin{aligned}
\text{h\_handshake} &= \text{MD5}(\text{handshake})||\text{SHA1}(\text{handshake})) \\
\text{MAC}_C &= \text{PRF}(\texttt{master\_secret}, \text{"}\texttt{client finished}\text{"}, \texttt{h\_handshake}) \\
\text{MAC}_S &= \text{PRF}(\texttt{master\_secret}, \text{"}\texttt{server finished}\text{"}, \texttt{h\_handshake}) \\
\text{master\_secret} &= \text{PRF}(\texttt{pre\_master\_secret}, \text{"}\texttt{master secret}\text{"}, \text{nonce}_C||\text{nonce}_S) \\
\text{key\_block} &= \text{PRF}(\texttt{master\_secret}, \text{"}\texttt{key expansion}\text{"}, \text{nonce}_S||\text{nonce}_C)
\end{aligned}
$$

handshake is the concatenation of all hanshake messages

MAC_C and MAC_S are of 12 bytes

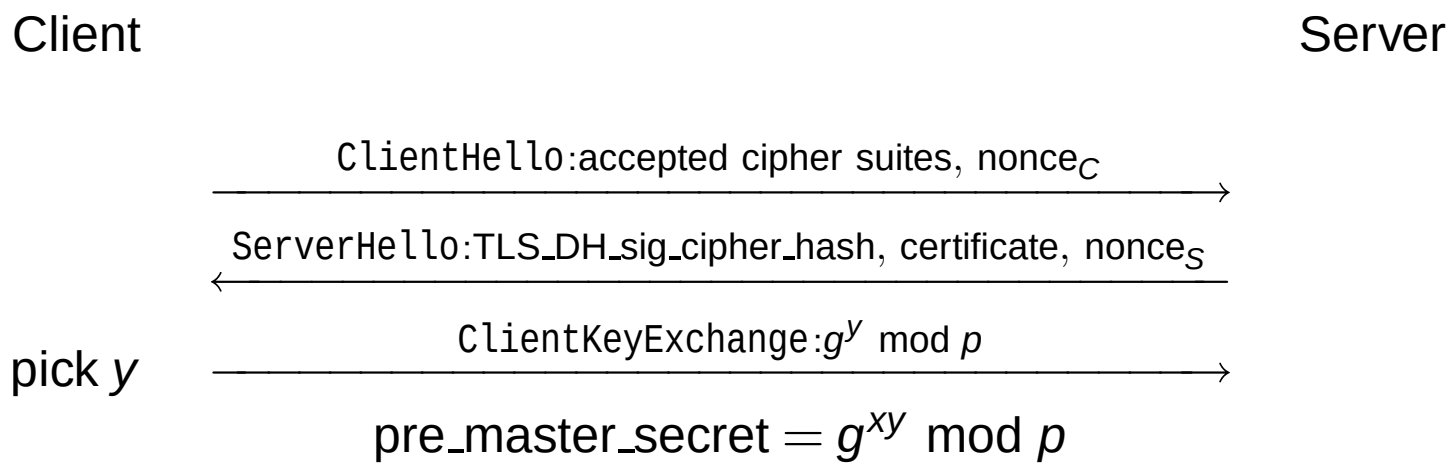
key_block is the concatenation of the four private keys and the two initial vectors.

RSA Key Exchange



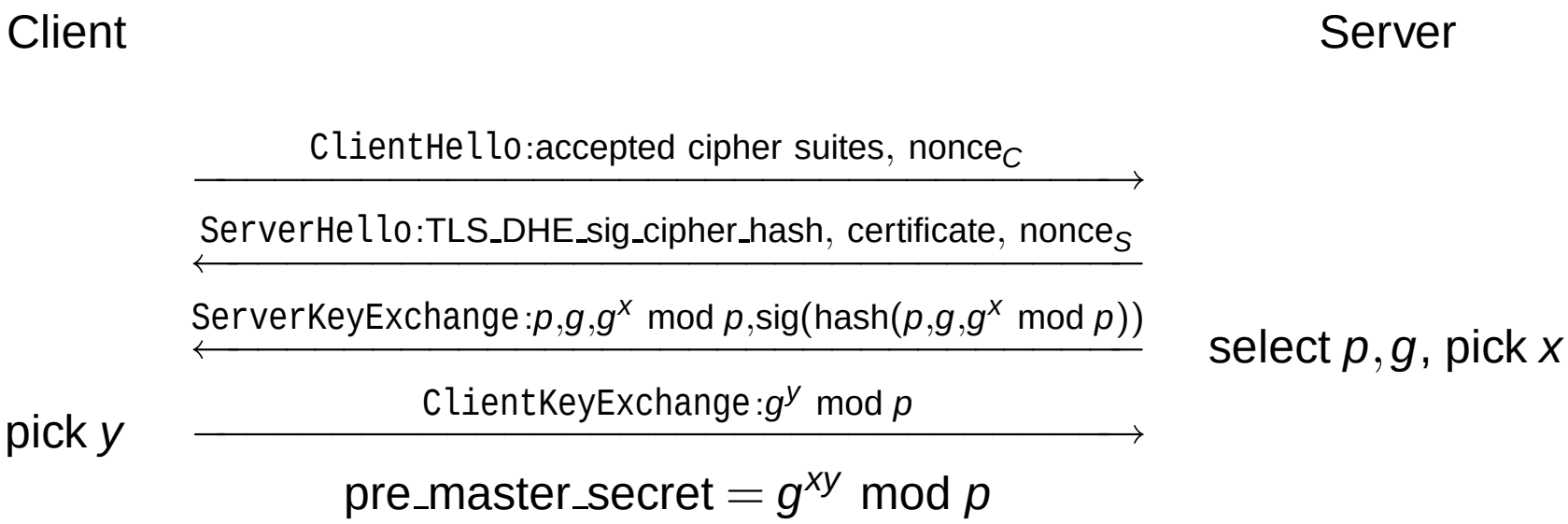
- RSA encryption is PKCS#1v1.5
- the RSA public key must be authenticated

DH_sig Key Exchange



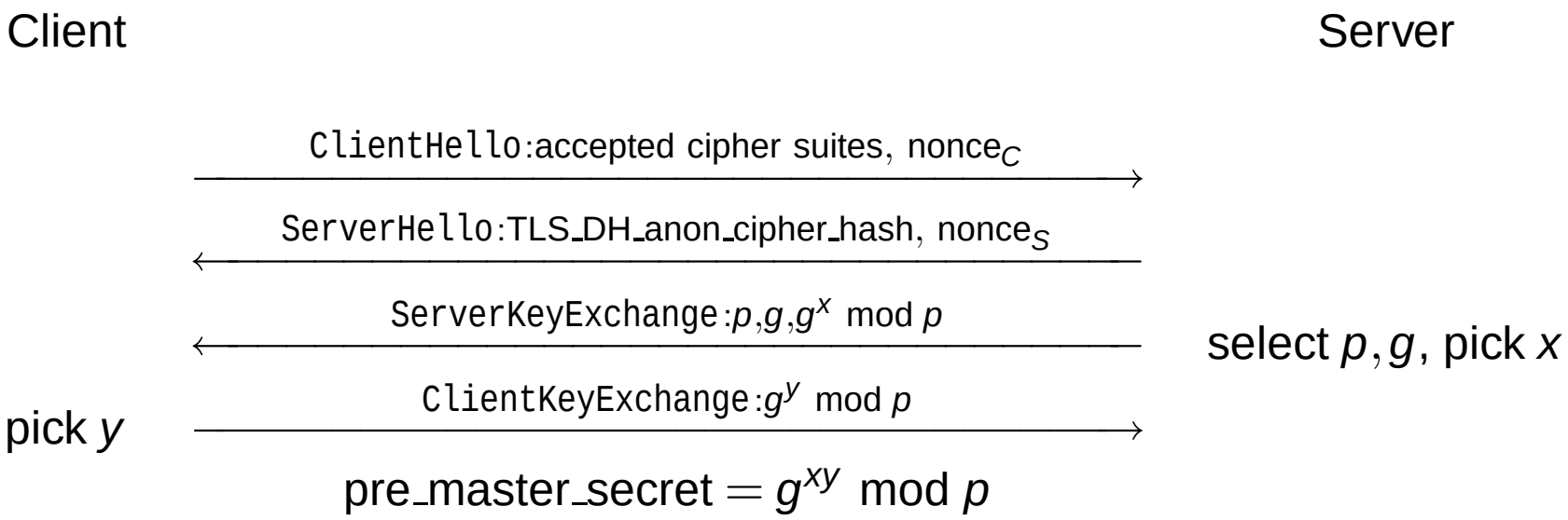
- the certificate is signed using sig algorithm
- the certificate includes $p, g, g^x \bmod p$
- this is fixed Diffie-Hellman where parameters are chosen by the server and server uses a fixed x

DHE_sig Key Exchange



- the sig public key must be authenticated in the certificate
- g^y mod p is not authenticated!

DH_anon Key Exchange



- Diffie-Hellman protocol is not authenticated!

Record Protocol

- split the application data into fragments of at most 2^{14} Bytes and send the fragments separately.
- (optional) compress the fragment
- append a MAC to the fragment
  The MAC is computed on a sequence number, the compression and TLS version materials, the compressed fragment.
- encrypt all this
- send this after a record header (type, version, length)

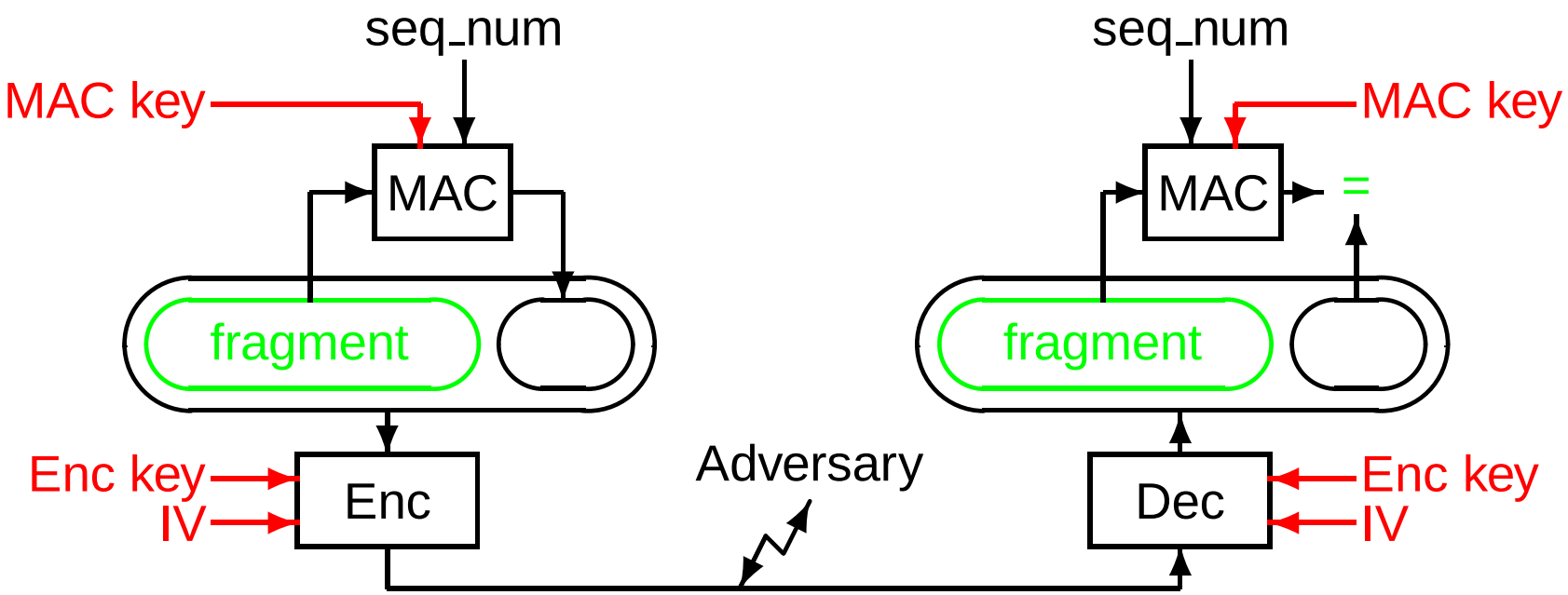
MAC in Record Protocol

More precisely the MAC of a fragment is computed as the HMAC with key `MAC_write_secret` on

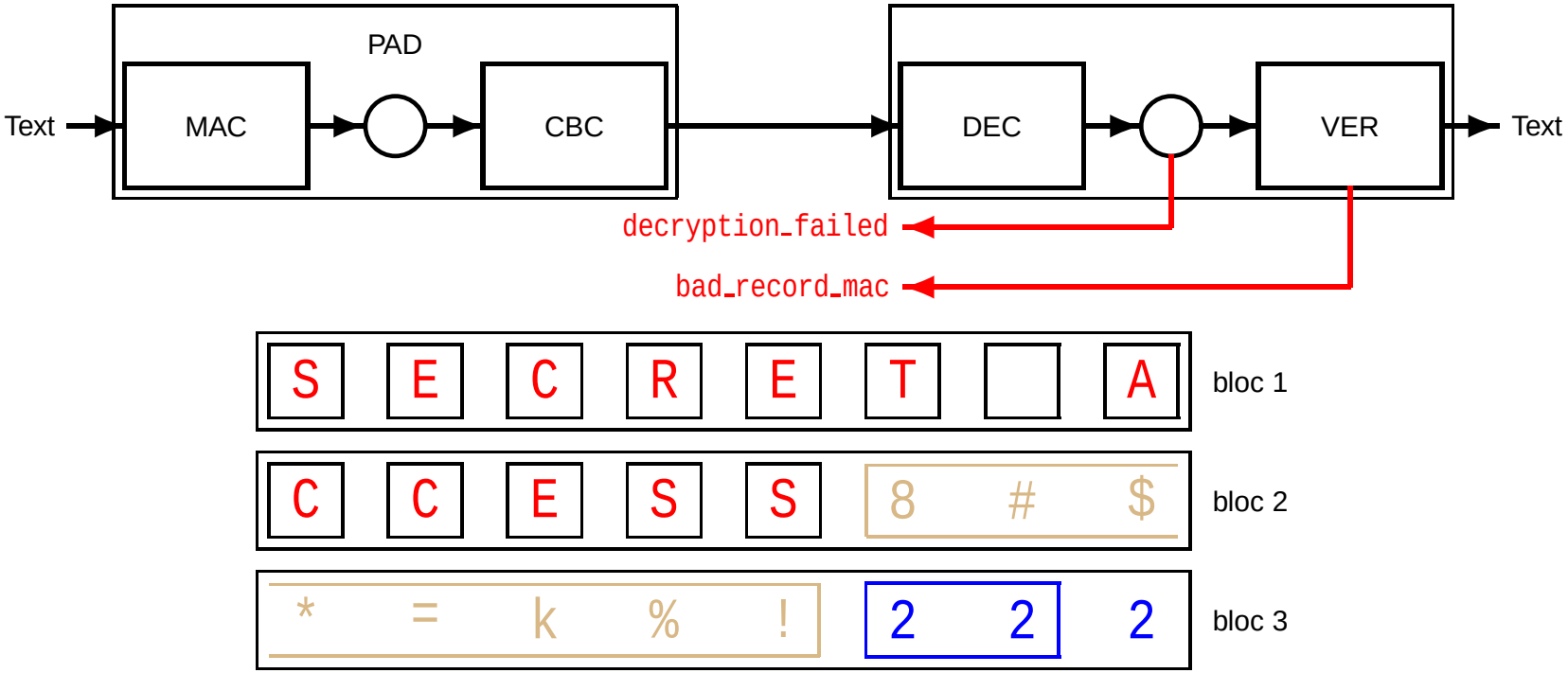
```
seq_num
TLSCompressed.type,TLSCompressed.version,TLSCompressed.length
TLSCompressed.fragment
```

- `MAC_write_secret` is the MAC key of the sender
- `seq_num` is the sequence number of the fragment
- `TLSCompressed.fragment` is the compressed fragment
- `TLSCompressed.length` is its actual length
- `TLSCompressed.type`
- `TLSCompressed.version` are some information about the TLS protocol (namely, the compression algorithm) that is being used

Secure Channel in SSL/TLS (Using CBC Encryption)



Using Block Ciphers in CBC Mode



Using Stream Ciphers

The RC4 stream cipher is used as a key-stream generator with one-time pad. The internal state of the generator is kept in the connection state so that the RC4 automaton continuously generates keystreams in order to encrypt the fragments sequence.

1 Secure Channels

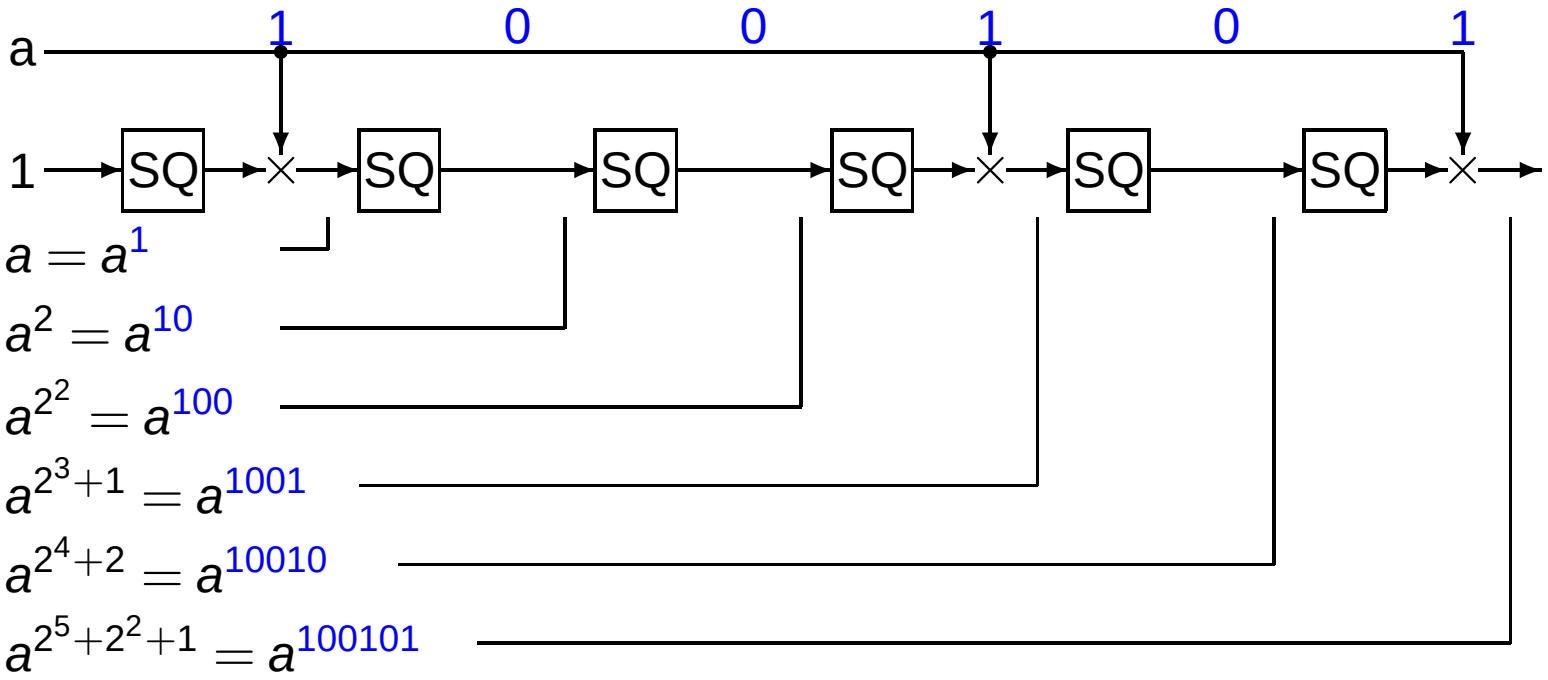
2 SSL/TLS

3 A Weakness in SSL/TLS

- Side Channel Attacks on RSA
- Weakness in PKCS Encryption
- Weakness in CBCPAD

Exponentiation From Left to Right

$$a^{100101} = a^{1\cdot 2^5+0\cdot 2^4+0\cdot 2^3+1\cdot 2^2+0\cdot 2^1+1\cdot 2^0}$$



Implementation

Input: a and n, two integers of at most ℓ bits, an integer e

Output: $x = a^e \bmod n$

Complexity: $\mathcal{O}(\ell^2 \log e)$

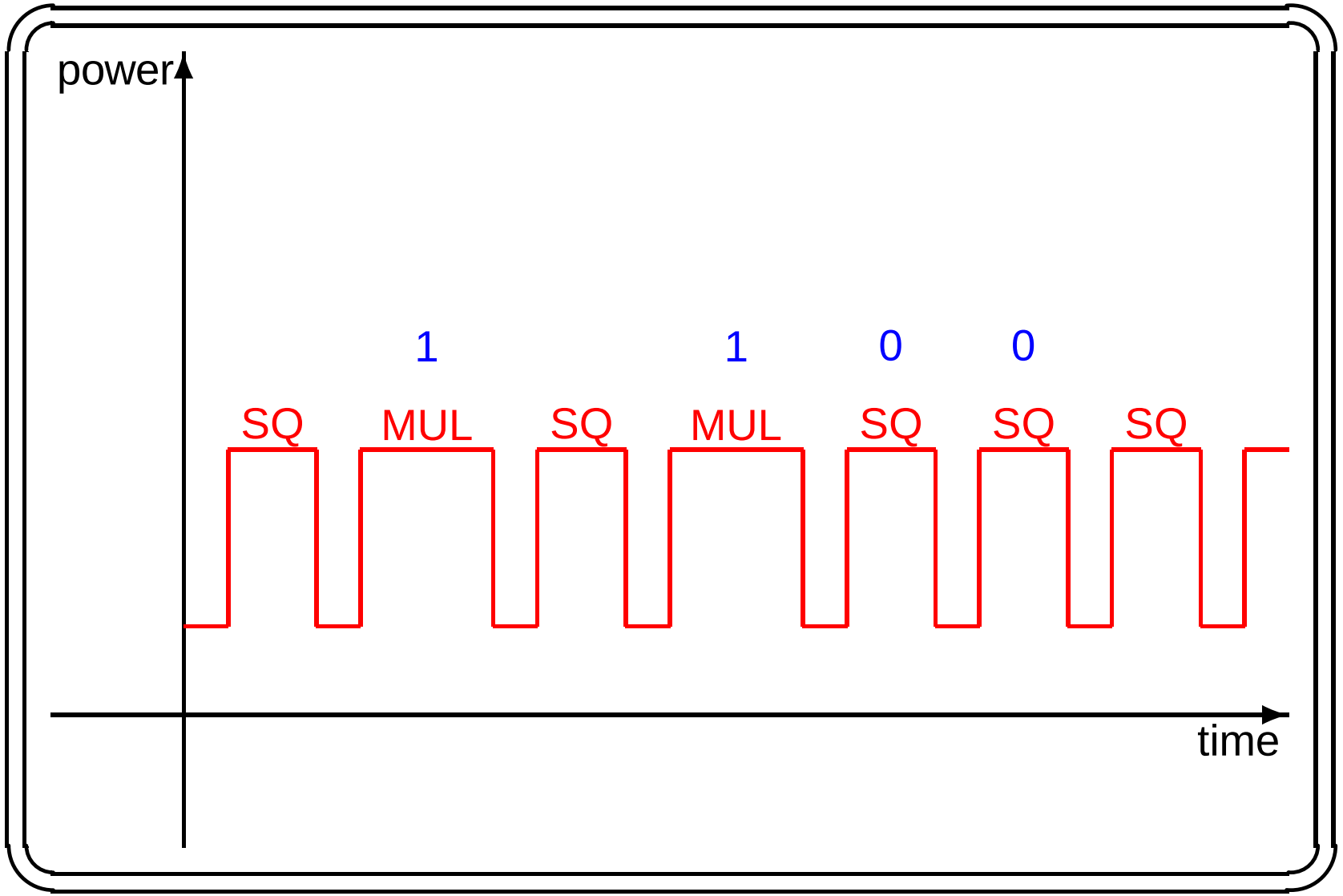
1: $x \leftarrow 1$
2: **for** $i = \ell - 1$ to 0 **do**
3: $\quad x \leftarrow x \times x \bmod n$ (SQ)
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad x \leftarrow x \times a \bmod n$ (MUL)
6: $\quad$ **end if**
7: **end for**

Power Analysis Attack

Computing $x = y^d \bmod N$ is performed by a device with external power supply by using the square-and-multiply algorithm.

- The power usage tells how what kind of operation is performed
- Cryptoprocessors have faster square than multiply algorithms
- The power usage tells when a square and a multiply is performed
- The attacker deduces d

SPA

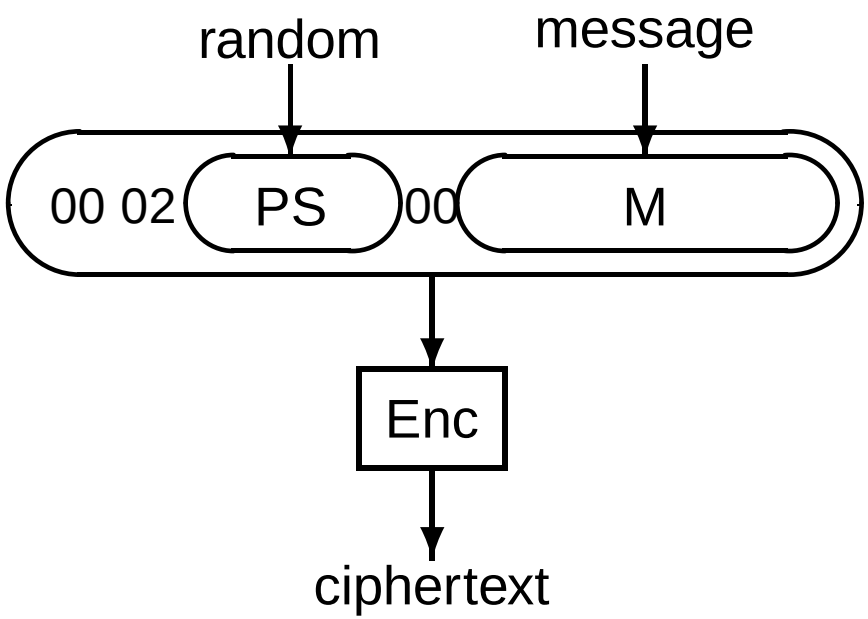


secret key is 1100...

Other Side Channel Attacks

- Differential fault analysis
- Timing attack
- Electromagnetic fields
- Noisy machines
- Cache attacks
- ...

PKCS#1v1.5 Encryption



Yet Another Side Channel Attack

Bleichenbacher's attack against PKCS#1v1.5:

- Attacker intercepts $y = x^e \bmod N$ and aims at recovering x
- Attacker plays with the server by sending fake ciphertexts y' of the form

$$y' = s^e y \bmod N$$

- Most of the time, y' does not decrypt well and the server issues an error message.
- If the server accepts, then $(y')^d \bmod n$ starts with 00 02, hence

$$2 \times 256^{k-2} \leq sx \bmod N < 3 \times 256^{k-2}$$

- By using this oracle 1 000 000 times, Attacker can reconstruct x

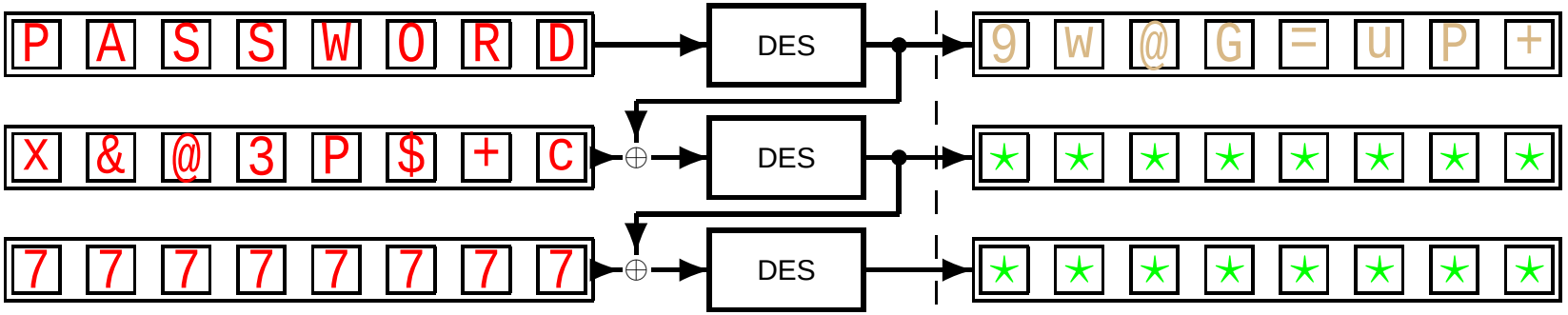
1 Secure Channels

2 SSL/TLS

3 A Weakness in SSL/TLS

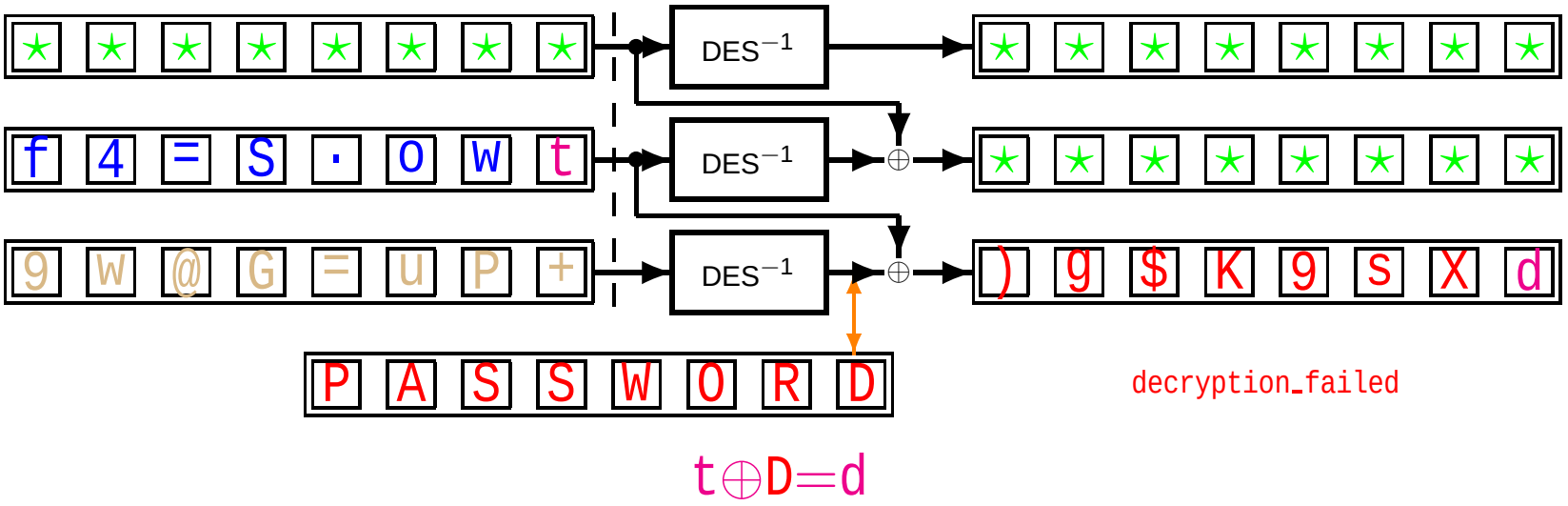
- Side Channel Attacks on RSA
- Weakness in PKCS Encryption
- Weakness in CBCPAD

CBCPAD Encryption

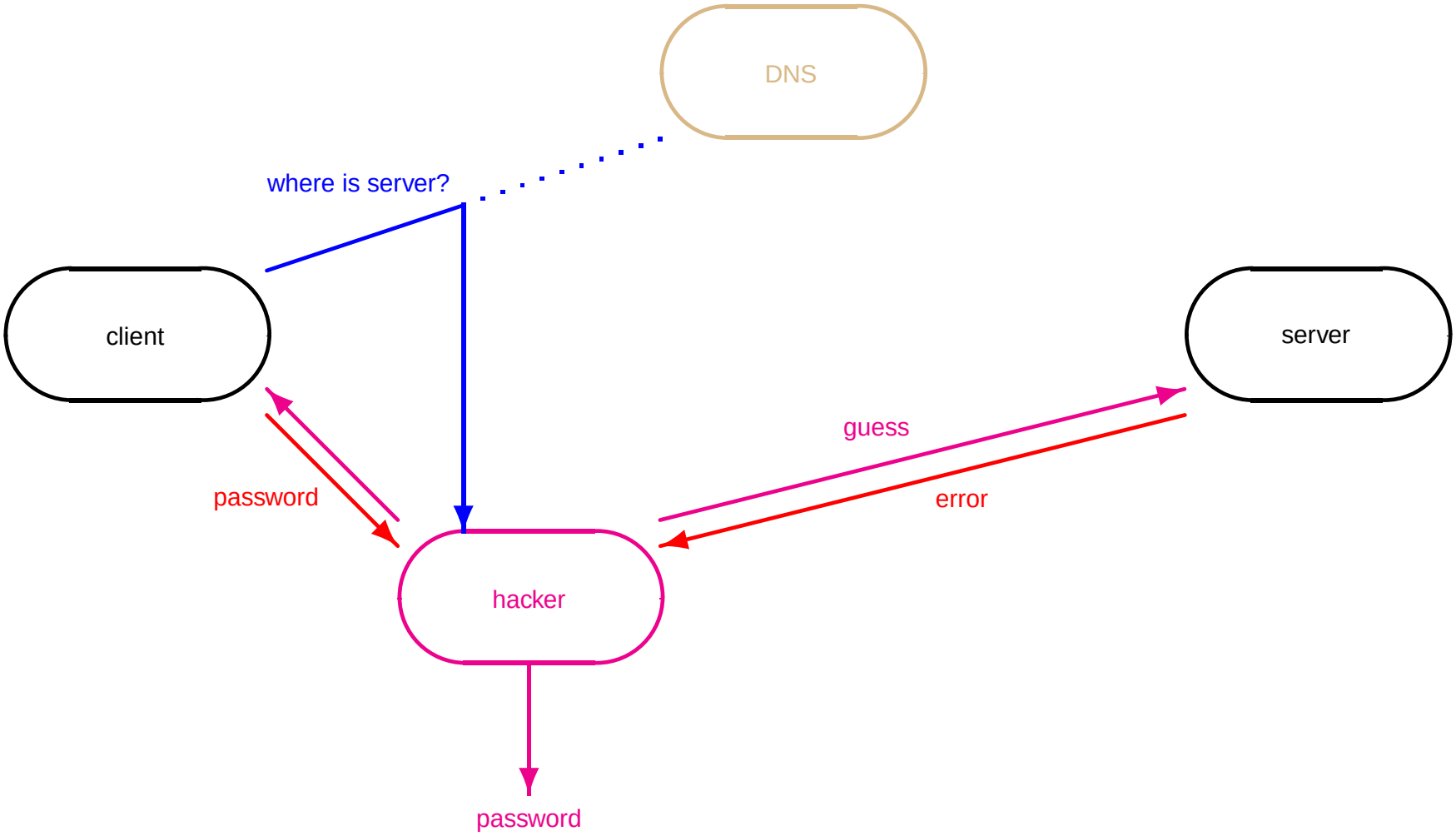


We would like to decrypt 9w@G=uP+

CBCPAD Decryption



The Attack Overview



Application to TLS: Two Problems

- Errors are sent encrypted through the Record Protocol
  $\longrightarrow$ errors are not available in clear

- Both types of errors are fatal alerts
  $\longrightarrow$ after one query the session is broken
  $\longrightarrow$ if the session restarts, it uses a fresh symmetric key

Transforming Peer into an Oracle

- we can ask “does $\mathrm{DES}^{-1}(y)$ end with byte c?”
- we can ask “does $\mathrm{DES}^{-1}(y)$ end with byte string u?”

- we can guess a byte within 128 queries on average
- we can compute $\mathrm{DES}^{-1}(y)$ within 1024 queries on average
- we can decrypt a message within $\ell \times 128$ queries on average

Implementation

DecryptBlock1(y)

1: **for** $i = 1$ to b **do**
2: $\quad c_i \leftarrow$**DecryptByte1**$(y, c_{i-1}||\ldots||c_1)$
3: **end for**
4: **return** $c_b||\ldots||c_1$

DecryptByte1(y,s)

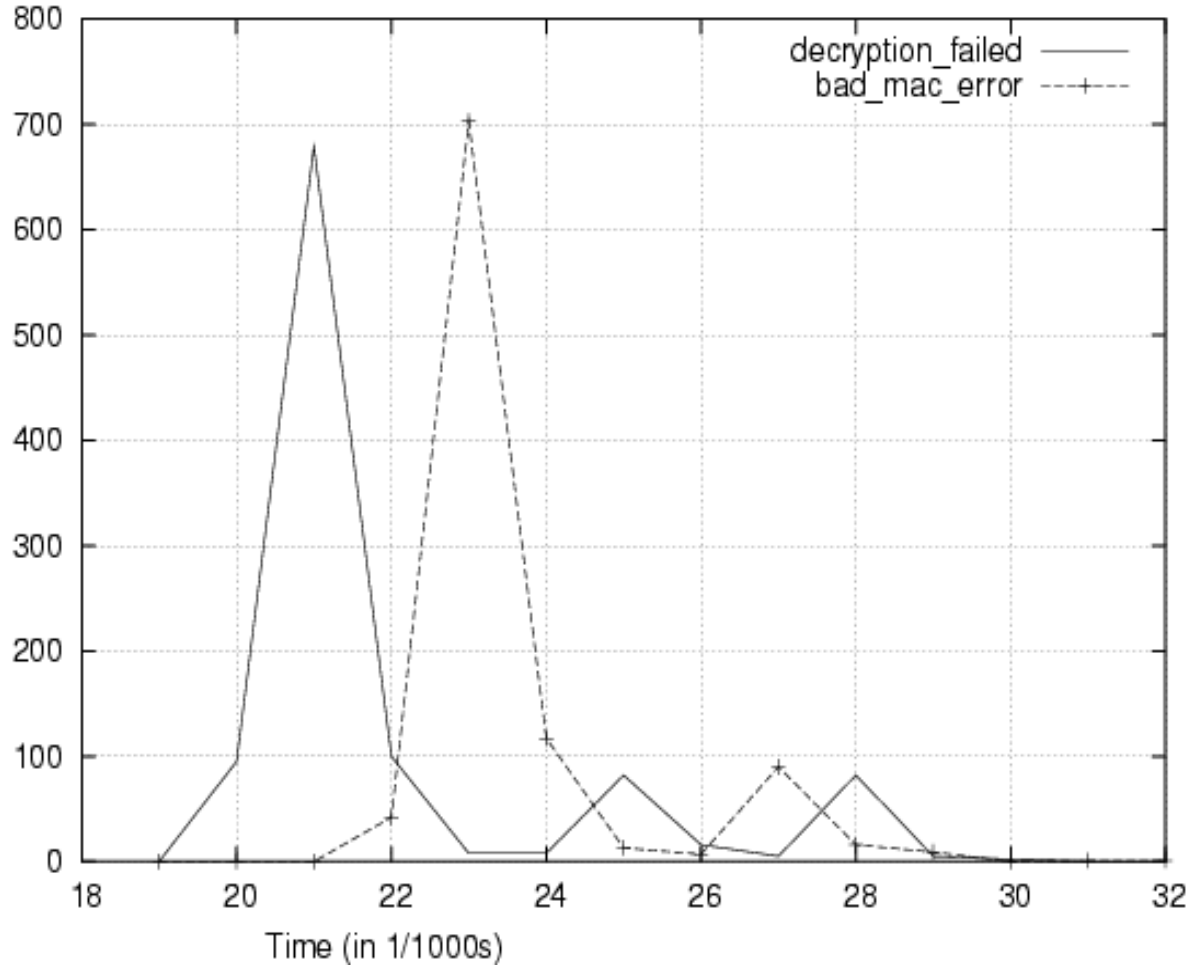
1: **for all** possible values of byte c **do**
2: $\quad$ **if Check1**$(y, c||s) = 1$ **then**
3: $\quad\quad$ **return** c
4: $\quad$ **end if**
5: **end for**

Check1(y,u)

1: let i be the length of u
2: let L be a random string of length $b-i$
3: let $R = (i-1)||(i-1)||\ldots||(i-1)$ of length i
4: $r \leftarrow L||(R \oplus u)$
5: build the fake ciphertext $r||y$ to be sent to the oracle
6: **return** $O(r||y)$

Using an Extra Side Channel

`bad_record_mac` errors take longer time to answer than `decryption_failed` errors



Regular Timing Attack

DecryptByte2(y, s)

1: **repeat**
2: **for all** possible values of byte c **do**
3: **if Check2**$(y, c||s) = 1$ **then**
4: **return** c
5: **end if**
6: **end for**
7: **until** byte is found

Check2(y, u)

1: make r in order to test u as in **Check1**
2: build the fake ciphertext $f||r||y$ to be sent to the oracle
(f is the longest possible random block sequence)
4: query the oracle n times and get $T_1, \ldots, T_n$
(answers which are larger than B are ignored)
6: **return** ACCEPT$(T_1, \ldots, T_n)$

$$\text{ACCEPT} : \frac{T_1 + \ldots + T_n}{n} > \tau'$$

Timing Attack with a Sequential Distinguisher

Check3(y,u)

1: make r in order to test u as in **Check1**
2: build the fake ciphertext $f||r||y$ to be sent to the oracle as in **Ckeck2**
3: $j \leftarrow 0$
4: **repeat**
5: $\quad j \leftarrow j+1$
6: $\quad$ query the oracle and get T_j
$\quad$ (a T_j larger than B is ignored and the query is repeated)
8: **until** $\mathrm{STOP}(T_1,\ldots,T_j)$
9: **return** $\mathrm{ACCEPT}(T_1,\ldots,T_j)$

$$\mathrm{STOP}: \quad T_1+\ldots+T_j-j\frac{\mu_R+\mu_W}{2} \notin [\tau'_-,\tau'_+]$$

$$\mathrm{ACCEPT}: \quad T_1+\ldots+T_j-j\frac{\mu_R+\mu_W}{2} > \tau'_+$$

Analysis

Theorem

Assuming that the two time responses have a normal distribution with the same standard deviation σ and expected values μ_R and μ_W, for some choices of τ'_+ and τ'_-, the above distinguisher is optimal in terms of probability of success vs complexity.

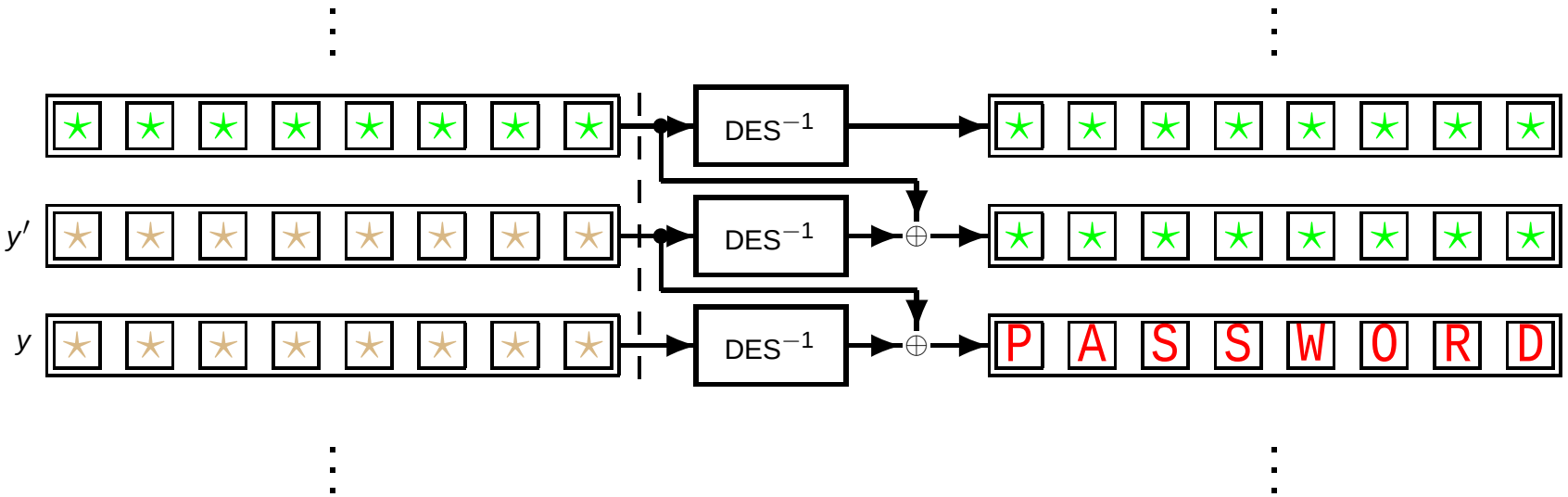
Thanks to the Wald Approximation, we can freely select ε_+ and ε_-, compute the corresponding τ'_+ and τ'_-, complexities and probability of success by

$$\tau'_+ \approx \frac{\sigma^2}{\mu_R - \mu_W} \log \frac{1-\varepsilon_-}{\varepsilon_+} \qquad J_W \approx -\frac{2\tau'_-}{\mu_R - \mu_W}$$

$$\tau'_- \approx \frac{\sigma^2}{\mu_R - \mu_W} \log \frac{\varepsilon_-}{1-\varepsilon_+} \qquad J_R \approx \frac{2\tau'_+}{\mu_R - \mu_W}$$

$$p \approx (1-\varepsilon_+)^{b\frac{|Z|-1}{2}} (1-\varepsilon_-)^b \qquad C = b\frac{|Z|-1}{2} J_W + bJ_R$$

Multi-Session Attack

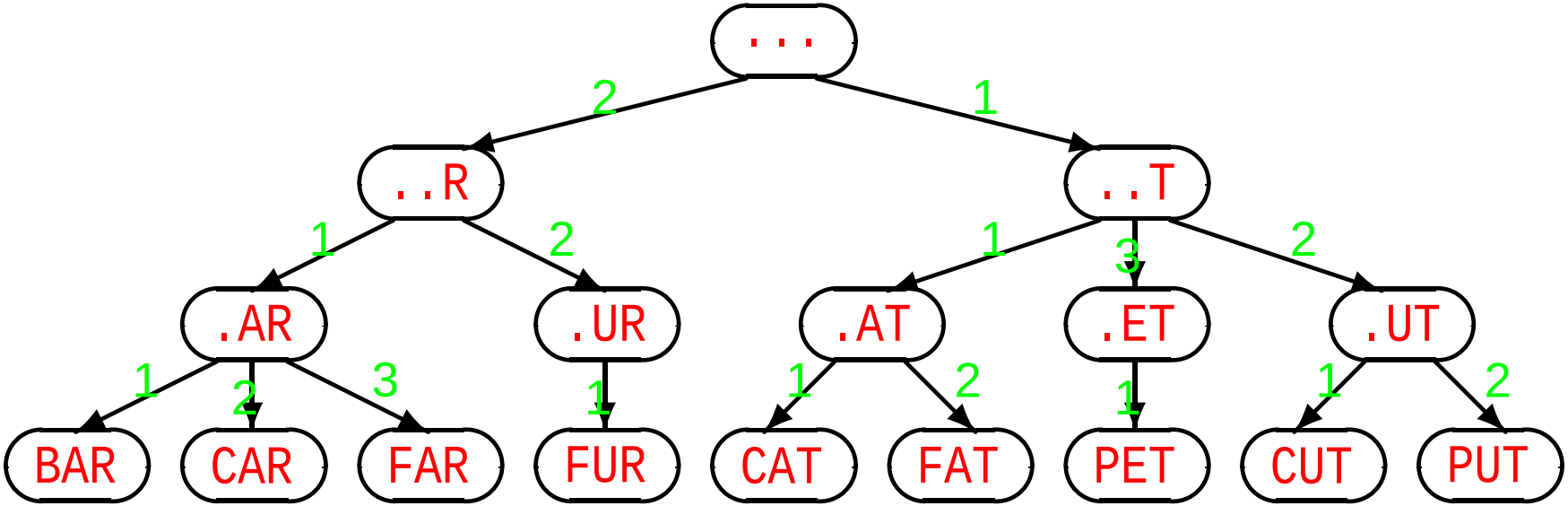


"does $\mathrm{DES}^{-1}(y)$ end with byte string u?"

$\Downarrow$

"does $\mathrm{DES}^{-1}(y) \oplus y'$ end with byte string u?"

Dictionary Attack



$$C_0 = \sum_{i=1}^{b} \sum_{c_i,\ldots,c_1} \Pr[c_i,\ldots,c_1] N(c_i \ldots c_1)$$

Analysis

$$
\begin{aligned}
C &\approx \frac{2\sigma^2}{(\mu_R - \mu_W)^2}\left(C_0 \log C_0 - (C_0 - 2b)\log\frac{C_0 - b}{b} - C_0 \log\log\frac{1}{p}\right) \\
\tau'_+ &\approx \frac{\sigma^2}{\mu_R - \mu_W}\left(\log C_0 + \log(C_0 - b) - \log b - \log\log\frac{1}{p}\right) \\
\tau'_- &\approx \frac{\sigma^2}{\mu_R - \mu_W}\left(\log(C_0 - b) - \log C_0 - \log b + \log\log\frac{1}{p}\right).
\end{aligned}
$$

Note that $p = 1 - e^{-\Omega(C)}$.

Final Attack

DecryptBlock4

1: **for** $i = 1$ to b **do**
2: $c_i \leftarrow$**DecryptByte4**$(c_{i-1}||\ldots||c_1)$
3: **end for**
4: **return** $c_b||\ldots||c_1$

DecryptByte4(s)

1: sort all possible c characters in order of decreasing likelihood.
2: **repeat**
3: **for all** possible values of character c **do**
4: **if Check4**$(c||s) = 1$ **then**
5: **return** c
6: **end if**
7: **end for**
8: **until** byte is found

Check4(u)

1: $j \leftarrow 0$
2: **repeat**
3: $j \leftarrow j + 1$
4: wait for a new session and get the current y and y' blocks
5: let i be the length of u
6: let L be a random string of length $b - i$
7: let $R = (i-1)||(i-1)||\ldots||(i-1)$ of length i
8: $r \leftarrow (L||(R \oplus u)) \oplus y'$
9: build the fake ciphertext $f||r||y$ to be sent to the oracle
(f is the longest possible random block sequence)
11: query the oracle and get T_j
(if it is larger than B then go back to Step 4)
13: **until** $\mathrm{STOP}(T_1, \ldots, T_j)$
14: **return** $\mathrm{ACCEPT}(T_1, \ldots, T_j)$

Application

We consider 4 scenarii for blocks of $b = 8$ characters

- random characters in an alphabet of 256 letters (full byte)
  $\longrightarrow \quad C_0 = 1028$
- random characters in an alphabet of 128 letters (ASCII character)
  $\longrightarrow \quad C_0 = 516$
- random characters in an alphabet of 64 letters (alphanumerical character)
  $\longrightarrow \quad C_0 = 260$
- block in a dictionary of $D = 712'786$ words
  $\longrightarrow \quad C_0 = 31$ (note that $\lceil \log_2 D \rceil = 20...$)

Numerical Values

Uniform distribution, $|Z| = 256$, $C_0 = 1028$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	4239	4750	5353	6139	7397	11335

Uniform distribution, $|Z| = 128$, $C_0 = 516$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	2179	2346	2738	3132	3764	5741

Uniform distribution, $|Z| = 64$, $C_0 = 260$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	1140	1269	1421	1620	1938	2934

Dictionary, $C_0 = 31$

p	0.5	0.6	0.7	0.8	0.9	0.99
C	166	181	199	223	261	380

Password Interception

- IMAP client: Outlook Express 6.x from Microsoft under Windows XP
- IMAP Rev 4 server
- Outlook checks (by default) for messages automatically every 5 minutes each folder created on the IMAP user account
- E.g. five folders (in, out, trash, read, and draft)
  $\longrightarrow$ 60 sessions every hour
- Outlook sends the login and password to the IMAP server using the following format:

  `XXXX␣LOG|IN␣"user|name"␣"p|assword"|<0x0d><0x0a><HMAC1><HMAC2>...`

  Here XXXX are four random digits which are incremented each time Outlook connects to the server.

Cipher Problem

- Outlook uses the RC4_MD5 algorithm by default (despite RFC2246 and RFC2595 suggest that 3DES_EDE_CBC_SHA should be supported by default).
- We had to force the IMAP server to only offer block ciphers in CBC mode.
- Other applications (e.g. stunnel) use block ciphers by default.

Format Problem

- It can be the case that the last bytes of the password belong to the first block of the MAC of the message.
- Example

  `|0021␣LOG|IN␣"name|"␣"passw|ord"<0x0d><0x0a><HMAC1><HMAC2>|...`
- It will not be possible to decrypt the last three characters from the password.

Conditions for a Successful Attack

- A critical piece of information is repeatedly encrypted at a predictable place.
- A block cipher in CBC mode is chosen.
- The attacker can sit in the middle and perform active attacks.
- The attacker can distinguish time differences between two types of errors.

Here we focused on the password access control in the IMAP protocol. We can also consider the basic authentication in HTTP which is also used for access control. This means that we can consider intercepting the password for accessing to an Intranet server.

Countermeasures

- The attack against WTLS was published in 2002.
- A countermeasure for TLS has been implemented in OpenSSL 0.9.6d and following versions:
  only the `bad_mac_error` error message is sent when an incorrect padding or an incorrect MAC are detected.
- This countermeasure is not enough because of timing attacks.
- A new countermeasure was implemented in OpenSSL 0.9.6i:
  we always check a MAC even if the padding is not correct.
- Other possible countermeasure:
  invert the padding and the MAC!

Lessons

- There are flaws, even in well esbablished standards
- We can make timing attacks over a network
- The order MAC-PAD-Encrypt should be reconsidered

Further Readings

- **Vaudenay**. *A Classical Introduction to Cryptography: Applications for Communication Security.* Springer. 2005.
  The lecture notes for my students
- **D. Bleichenbacher**.
  Chosen Ciphertext Attack Against Protocols Based on the RSA Encryption Standard PKCS#1.
  In *Advances in Cryptology (CRYPTO'98)*, LNCS vol. 1462, pp. 1–12, 1998.
- **B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux**.
  Password Interception in a SSL/TLS Channel.
  In *Advances in Cryptology (CRYPTO'03)*, LNCS vol. 2729, pp. 583–599, 2003.

Q & A