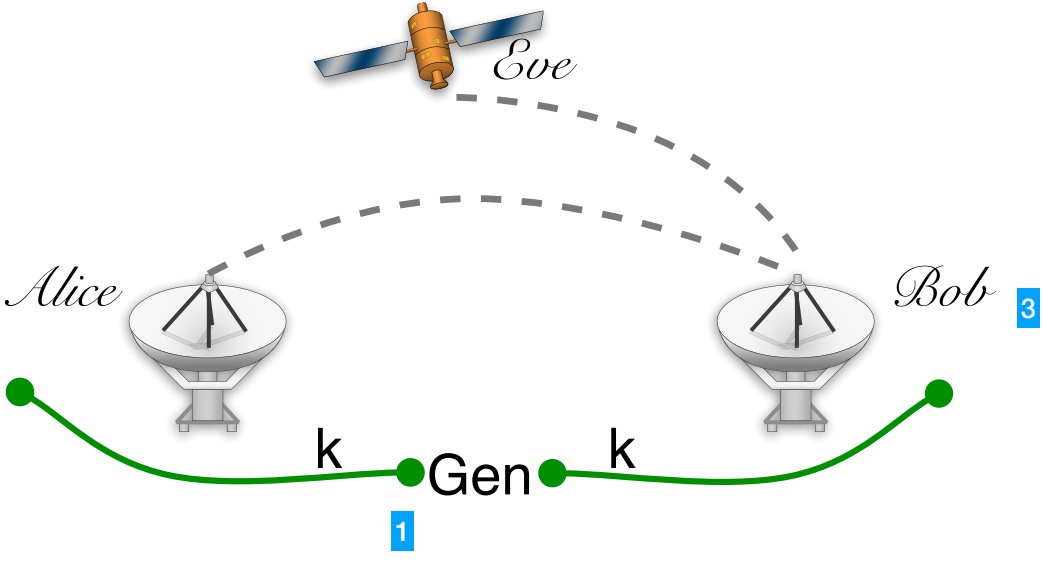
2550 Intro to cybersecurity

L7: Crypto: MACs, PRF, PKC

abhi shelat

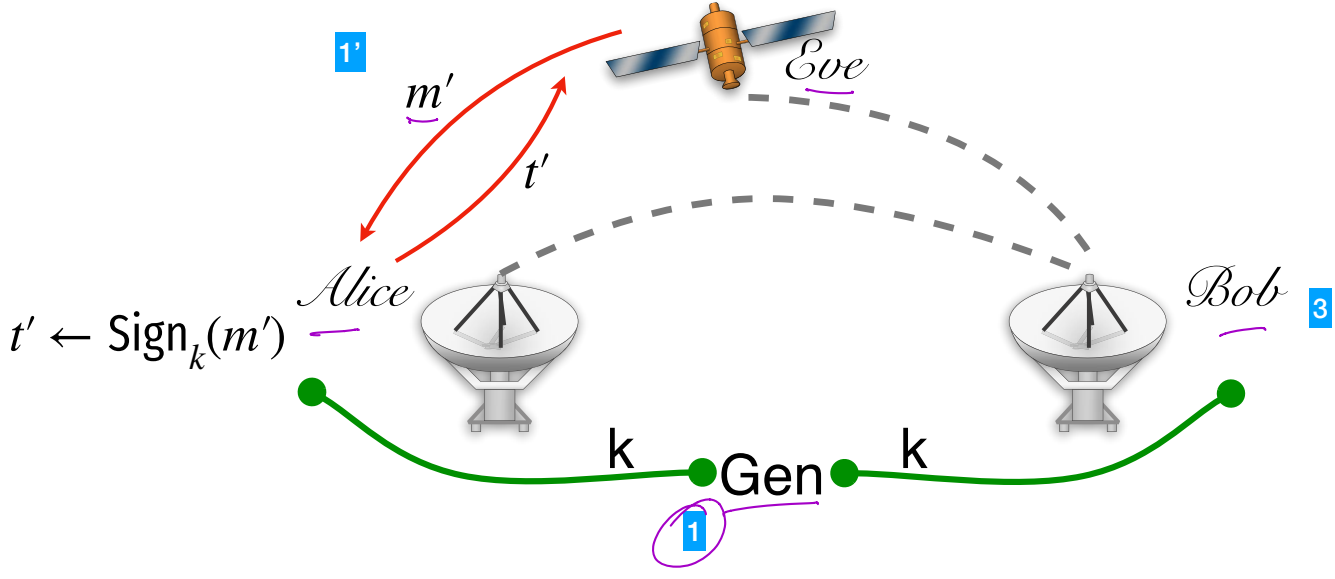
Secure MAC

"even when given a tag oracle,
an adversary cannot forge a tag for
any message of its choosing "



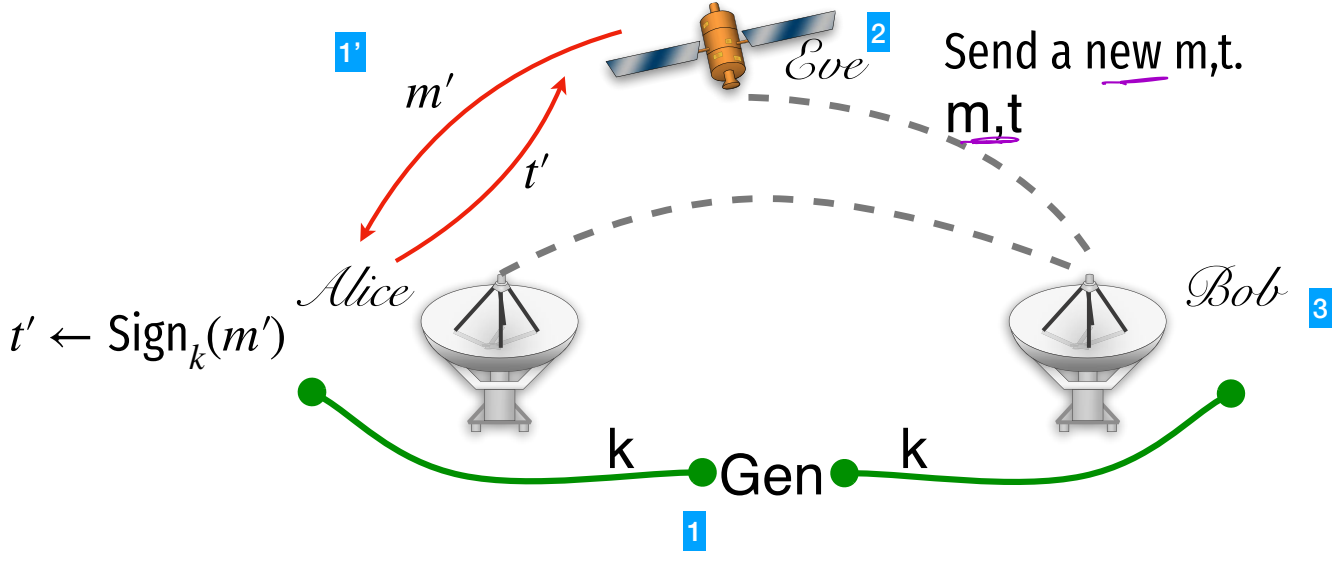
Secure MAC

"even when given a tag oracle,
an adversary cannot forge a tag for
any message of its choosing "



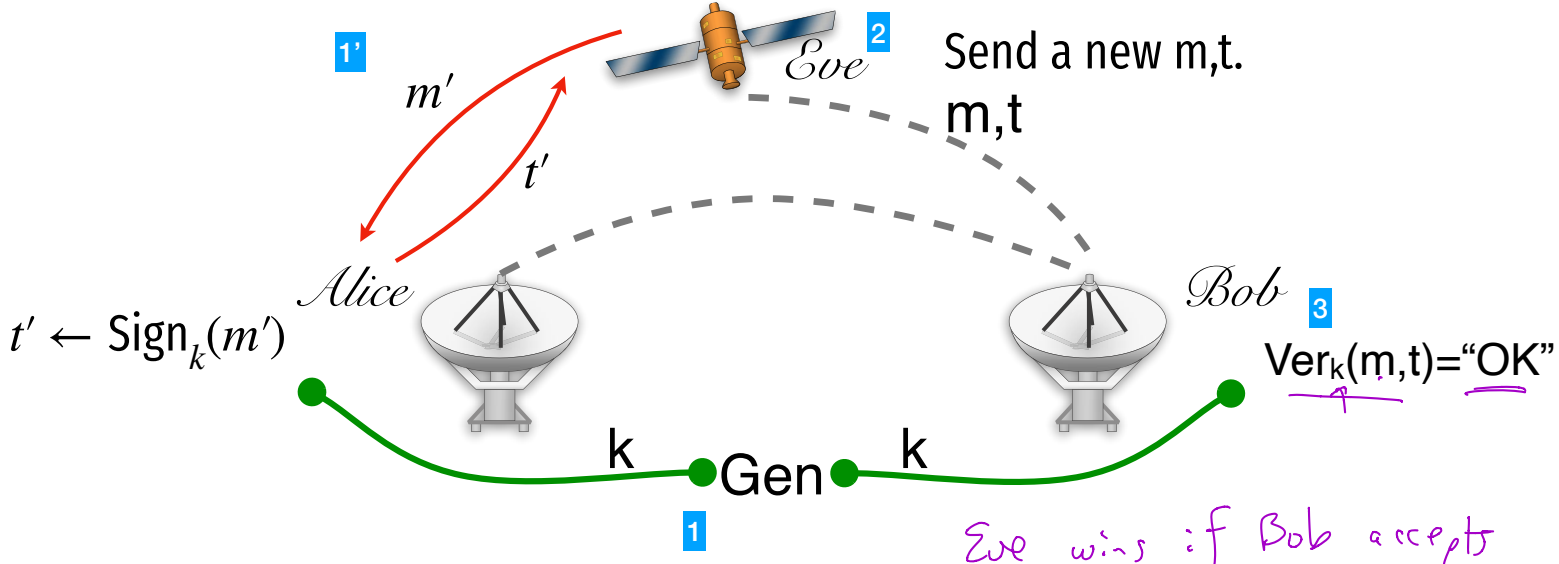
Secure MAC

"even when given a tag oracle,
an adversary cannot forge a tag for
any message of its choosing "



Secure MAC

"even when given a tag oracle, an adversary cannot forge a tag for any message of its choosing "



How can we implement a secure MAC?

Random functions

$$R : \{0,1\}^n \rightarrow \{0,1\}^n$$

128 bits

(length preserving)

In	Out
0	010...00
1	110...110
2	001..100
3	110...001
...	
2^n-1	100...111

Each entry is just randomly chosen.

How can you describe a random function?

$$R : \{0,1\}^n \rightarrow \{0,1\}^n$$

In	Out
0	010...00
1	110...110
2	001..100
3	110...001
...	
2^n-1	100...111

this part of the table describes the function.

$n \cdot 2^n$ size.

How can you describe a random function?

$R : \{0,1\}^n \rightarrow \{0,1\}^n$

In	Out
0	010...00
1	110...110
2	001..100
3	110...001
...	
2^n-1	100...111

total space of table:

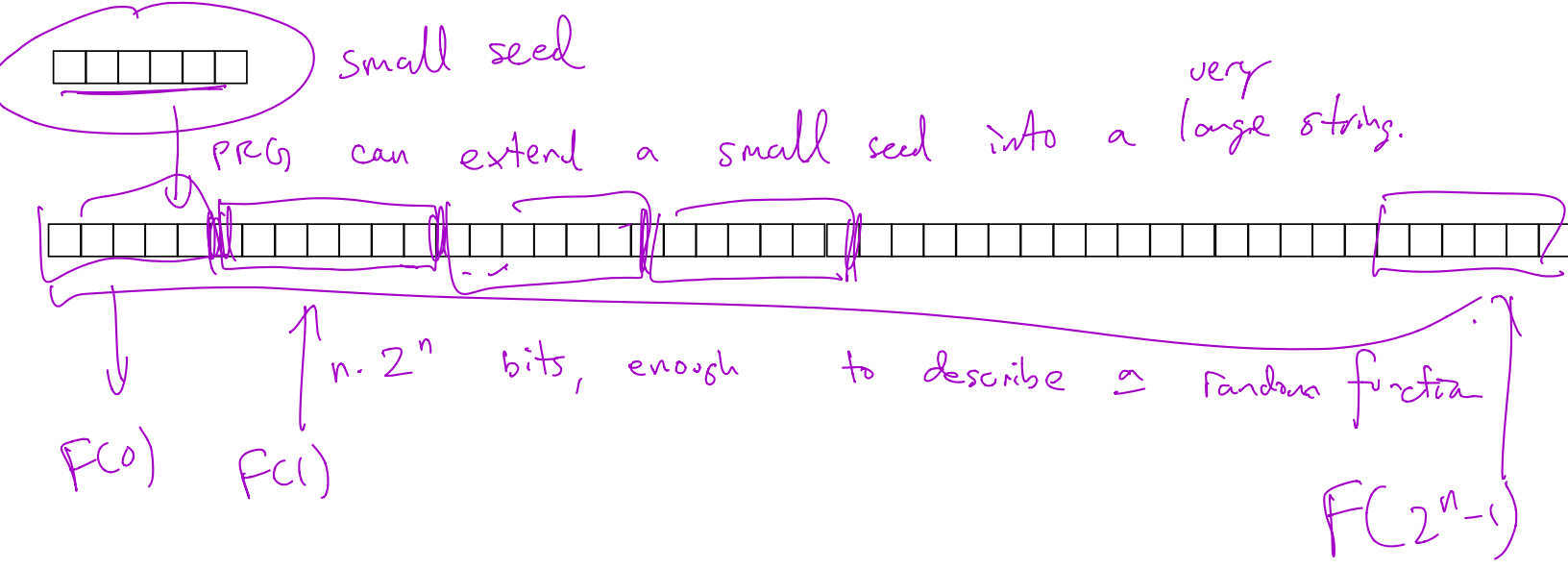
$n \cdot 2^n$ bits

How can you describe a random function?

$R : \{0,1\}^n \rightarrow \{0,1\}^n$

In	Out
0	010...00
1	110...110
2	001..100
3	110...001
...	
2^n-1	100...111

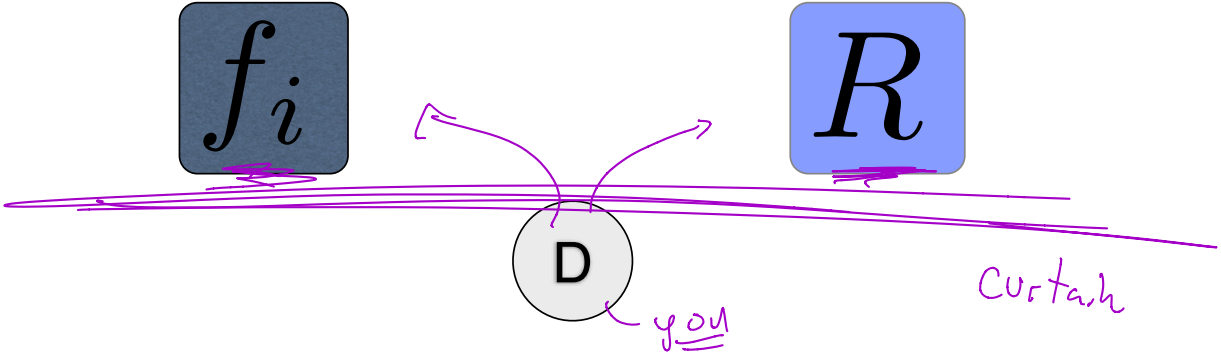
total space of table: $2^n n$


small seed
PRG can extend a small seed into a very large string.
$n \cdot 2^n$ bits, enough to describe a random function
F(0)
F(1)
F(2^n - 1)

Defining pseudo-random functions

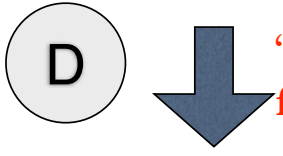
a function family that is comp. indistinguishable from a random function

Defining a prf



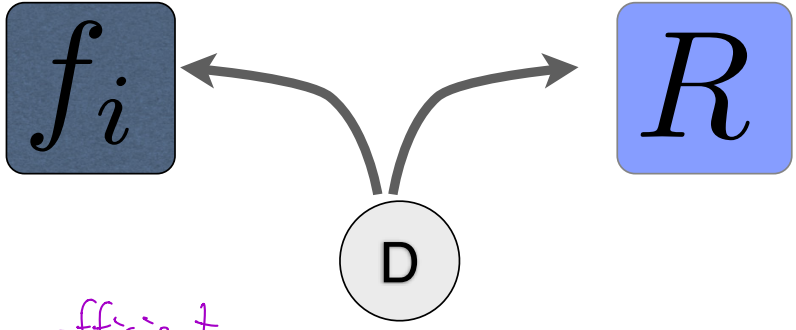
distinguisher is given one
of these two functions
and must guess which

Defining a prf



distinguisher is given one
of these two functions
and must guess which

Defining a prf

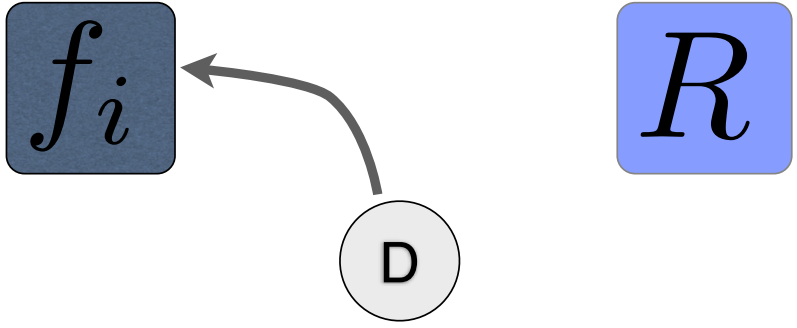


efficient

distinguisher is given oracle access to one of these two functions and must guess which

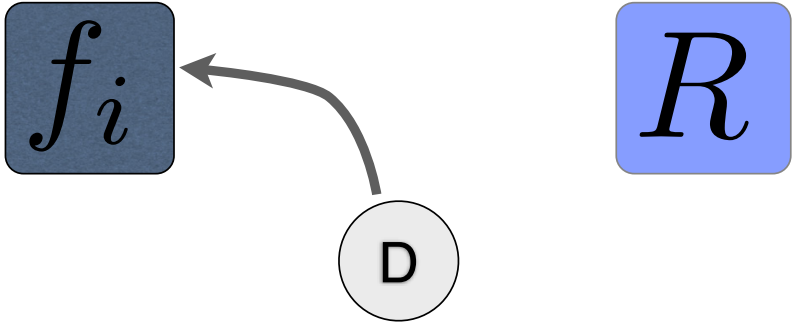
$$\Pr[K \leftarrow Gen : D^{f_K(\cdot)} = 1] - \Pr[K \leftarrow Gen(n) : D^{RF(\cdot)} = 1] \leq \epsilon(n)$$

Defining a prf



distinguisher is given oracle
access to one of these two functions
and must guess which

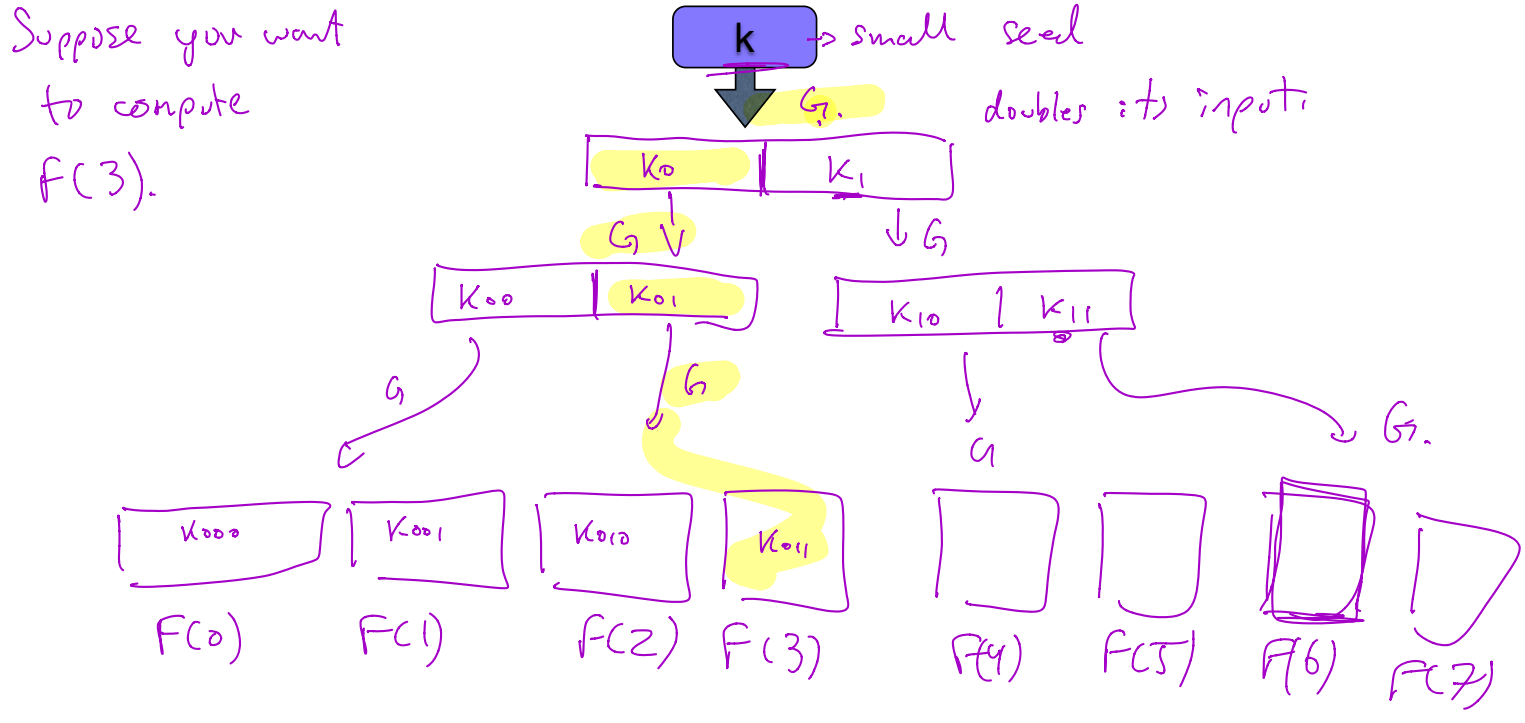
Defining a prf



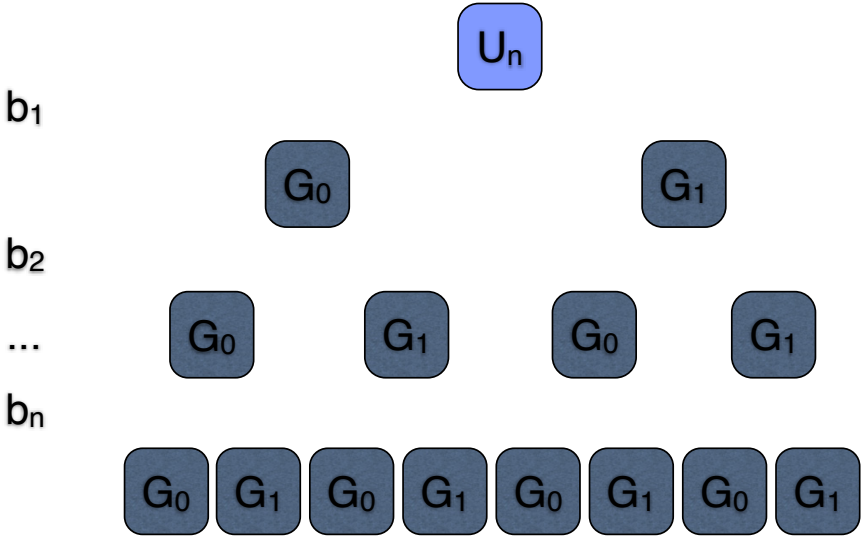
distinguisher is given oracle
access to one of these two functions
and must guess which

$$\Pr[f_k \leftarrow F_n;\ D^{f_k}(1^n) = 1] - \Pr[R \leftarrow RF_n;\ D^{R}(1^n) = 1] < \epsilon(n)$$

How to construct a PRF from a PRG

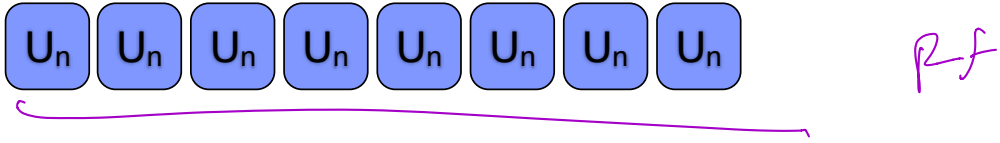


PRF

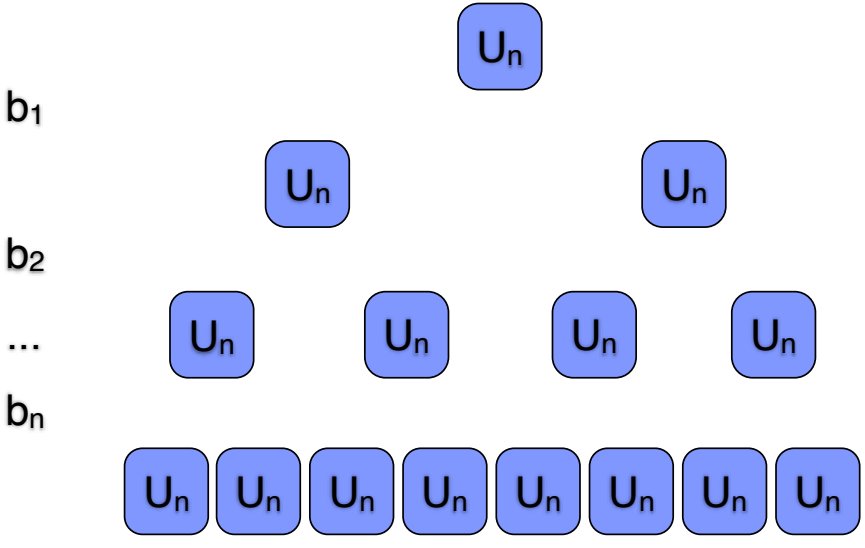


← if there are 2^n possible outputs, i.e. leaves, then it need n PRG calls.

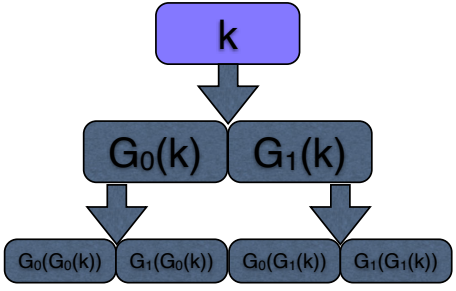
Random function



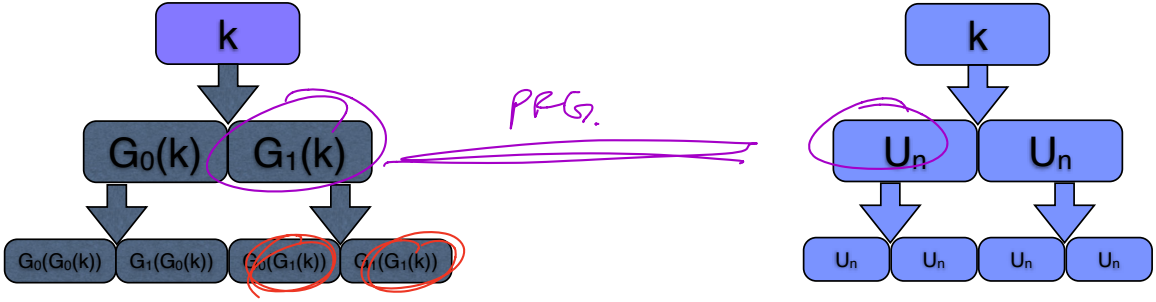
Random function



Why is this secure?

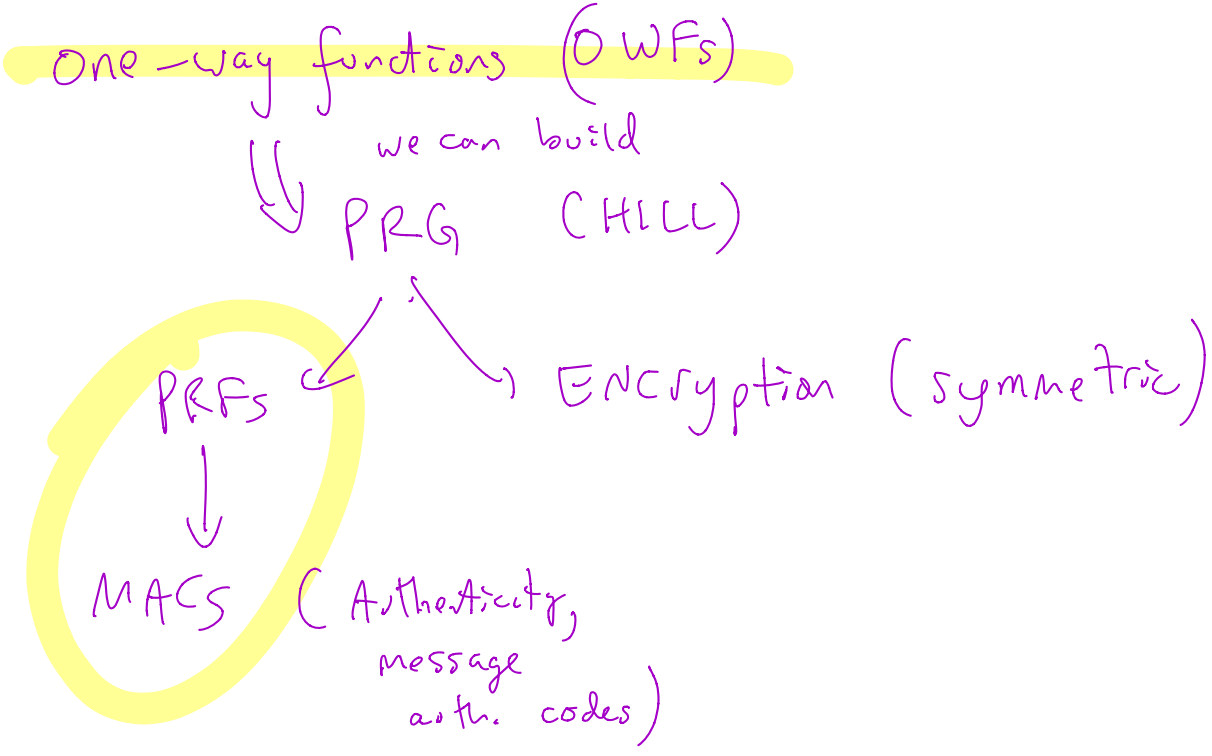


Why is this secure?



"hybrid argument"

TOOLS we've covered so far:



Construction of a MAC

Let F be a PRF.

Gen(1^n): $k \leftarrow \{0,1\}^n$ uniformly

Sign$_k$(m): $t \leftarrow F_k(m)$

Ver$_k$(m,t): check that $t = F_k(m)$

"Apply the PRF to the message"

Construction of a MAC

let $\{F_k\}$ be a PRF family

Gen(1^n):

$\text{Sign}_k(m)$:

$\text{Ver}_k(m,t)$:

Construction of a MAC

let $\{F_k\}$ be a PRF family

Gen(1^n): $k \leftarrow U_n$

$\text{Sign}_k(m)$:

$\text{Ver}_k(m,t)$:

Construction of a MAC

let $\{F_k\}$ be a PRF family

Gen(1^n): $k \leftarrow U_n$

Sign$_k$(m): $t \leftarrow F_k(m)$

Ver$_k$(m,t):

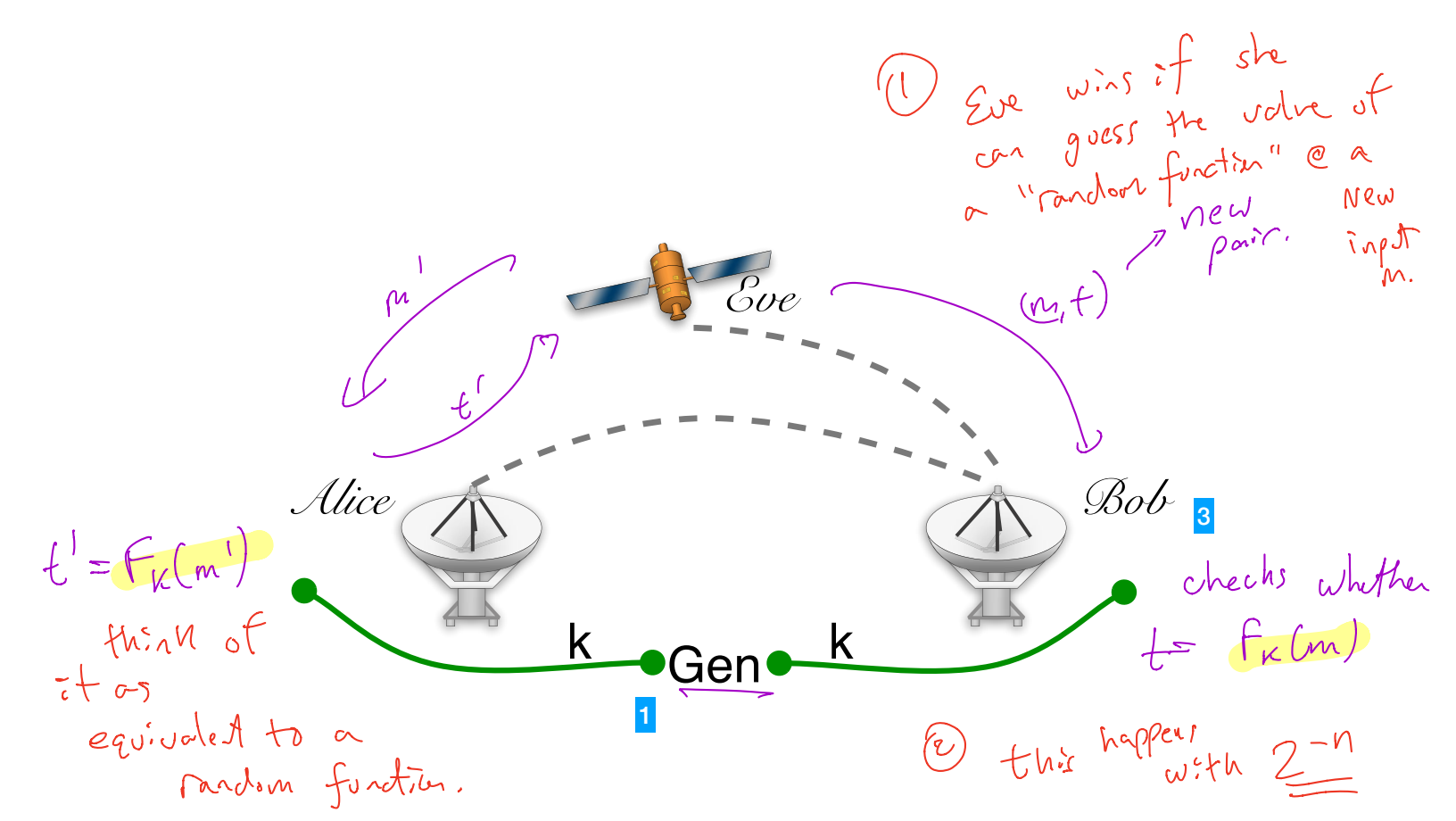
Construction of a MAC

let $\{F_k\}$ be a PRF family

Gen(1^n): $k \leftarrow U_n$

Sign$_k$(m): $t \leftarrow F_k(m)$

Ver$_k$(m,t): accept if $t \stackrel{?}{=} F_k(m)$



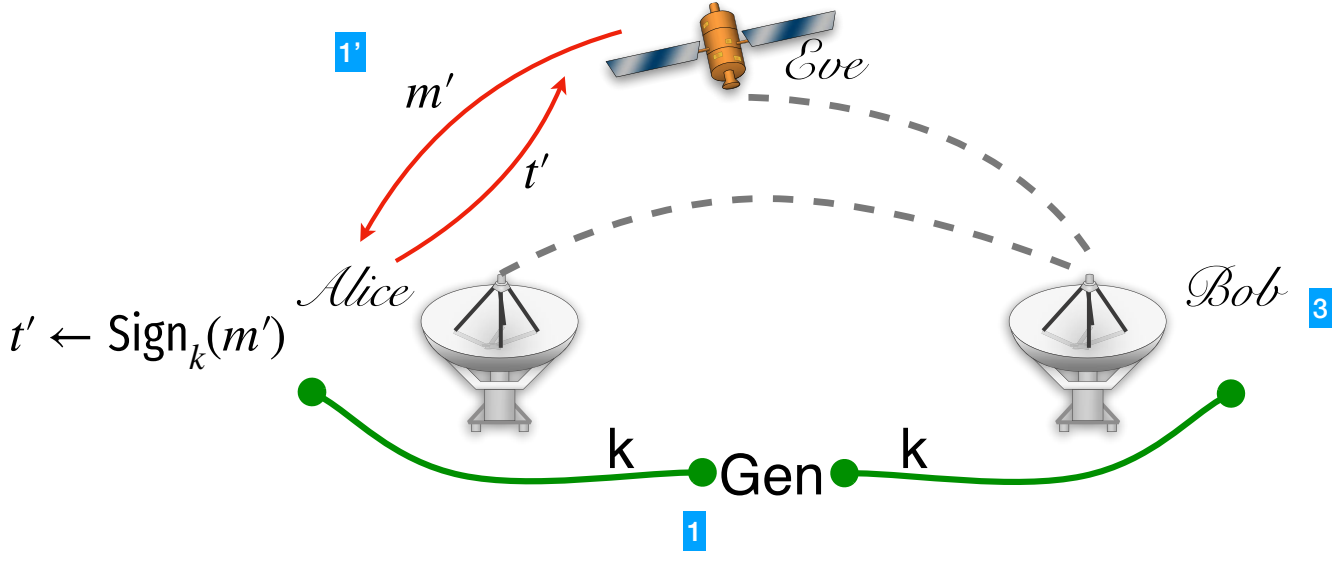
① Eve wins if she can guess the value of a "random function" @ a new pair. new input m.

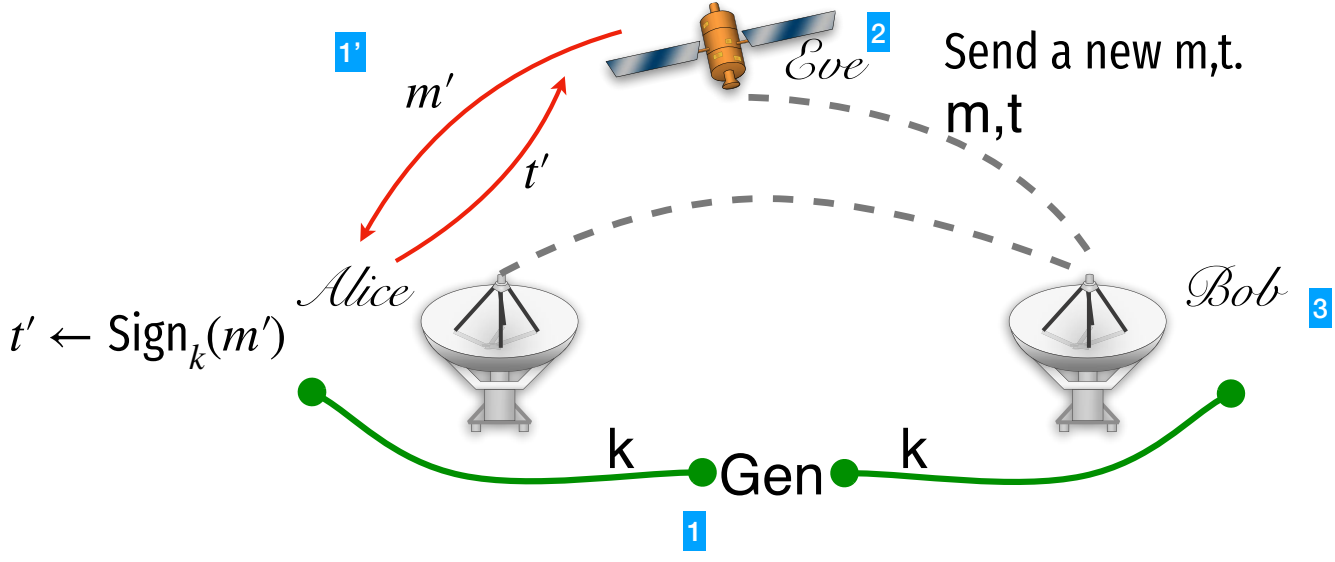
$t' = F_k(m')$

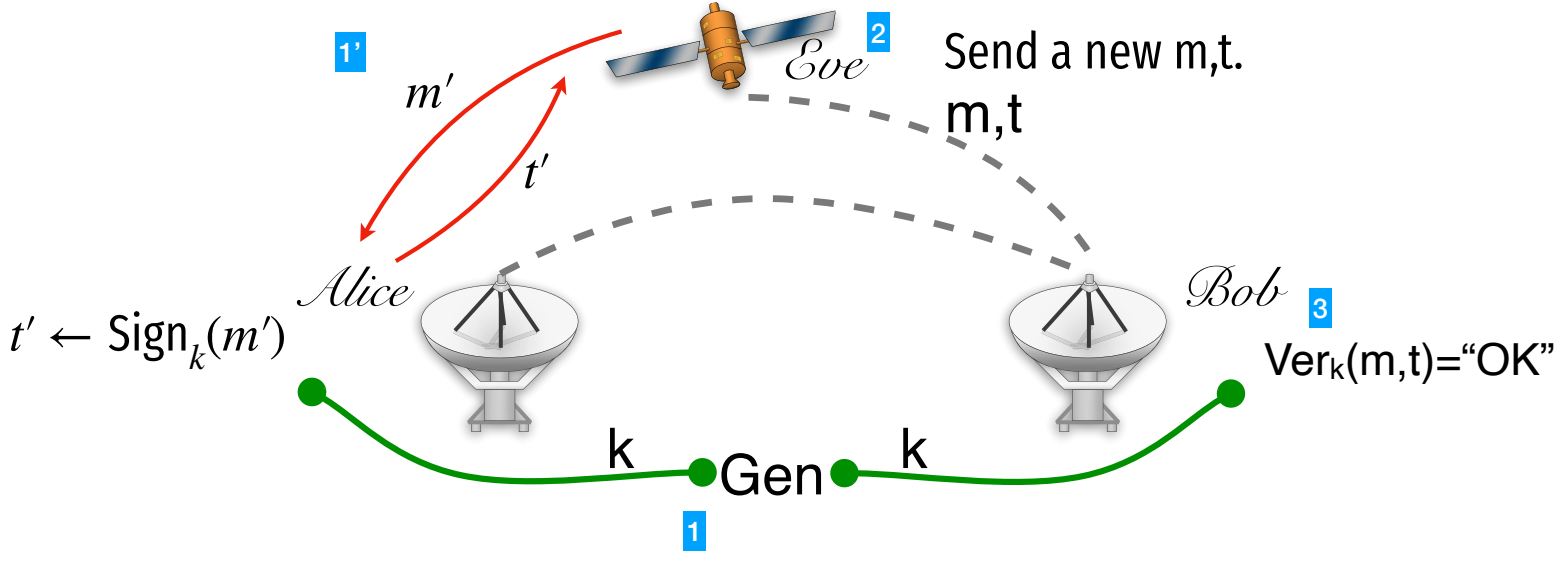
think of it as equivalent to a random function.

checks whether $t = F_k(m)$

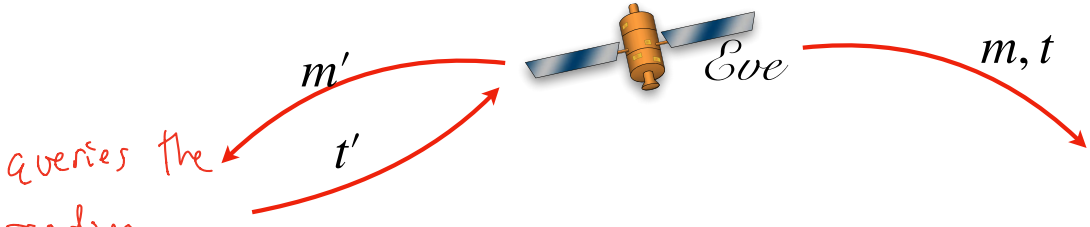
② this happens with 2^{-n}


1'
m'
t'
Eve
Alice
Bob
3
t' ← Sign_k(m')
k
Gen
k
1


1'
m′
t′
Eve
2
Send a new m,t.
m,t
Alice
Bob
3
t′ ← Sign_k(m′)
k
Gen
k
1


1'
m′
t′
Eve
2
Send a new m,t.
m,t
Alice
Bob
3
t′ ← Sign_k(m′)
Ver_k(m,t)="OK"
k
Gen
k
1

Security intuition



queries the random function @ several values

but has to guess the value @ a new location.

$$\Pr[F_k(m) = t] = 2^{-n}$$

Security proof idea

let $\{F_k\}$ be a prf family

Gen(1^n): $k \leftarrow U_n$ Tag$_k$(m): $t \leftarrow F_k(m)$

Security proof idea

let $\{F_k\}$ be a prf family

Gen(1^n): $k \leftarrow U_n$ Tag$_k$(m): $t \leftarrow F_k(m)$

$$\Pr\left[\begin{array}{l} k \leftarrow Gen(1^n); (m,t) \leftarrow A^{Tag_k(\cdot)} : \\ \quad Ver_k(m,t) = 1 \\ \quad \text{and A didn't query m} \end{array}\right]$$

Security proof idea

let $\{F_k\}$ be a prf family

Gen(1^n): $k \leftarrow U_n$ Tag$_k$(m): $t \leftarrow F_k(m)$

$$\Pr\left[\begin{array}{l} k \leftarrow Gen(1^n); (m,t) \leftarrow A^{Tag_k(\cdot)} : \\ \quad Ver_k(m,t) = 1 \\ \quad \text{and A didn't query m} \end{array}\right]$$

$$< \mu(n) + \Pr\left[\begin{array}{l} k \leftarrow Gen(1^n); (m,t) \leftarrow A^{RF_n(\cdot)} : \\ \quad Ver_k(m,t) = 1 \\ \quad \text{and A didn't query m} \end{array}\right]$$

Security proof idea

let $\{F_k\}$ be a prf family

Gen(1^n): $k \leftarrow U_n$ Tag$_k$(m): $t \leftarrow F_k(m)$

$$\Pr\left[\begin{array}{l} k \leftarrow Gen(1^n); (m,t) \leftarrow A^{Tag_k(\cdot)} : \\ \quad Ver_k(m,t) = 1 \\ \quad \text{and A didn't query m} \end{array}\right]$$

$$< \mu(n) + \Pr\left[\begin{array}{l} k \leftarrow Gen(1^n); (m,t) \leftarrow A^{RF_n(\cdot)} : \\ \quad Ver_k(m,t) = 1 \\ \quad \text{and A didn't query m} \end{array}\right]$$

$$< \mu(n) + 2^{-n}$$

Efficiency of PRF constructions

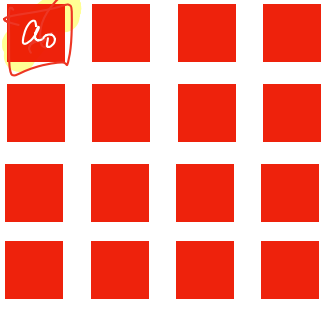
① BM PRG: → 1 exponentiation per bit of output.

AES(k,m)

substitution-permutation network

① expand

② writes the msg into this form

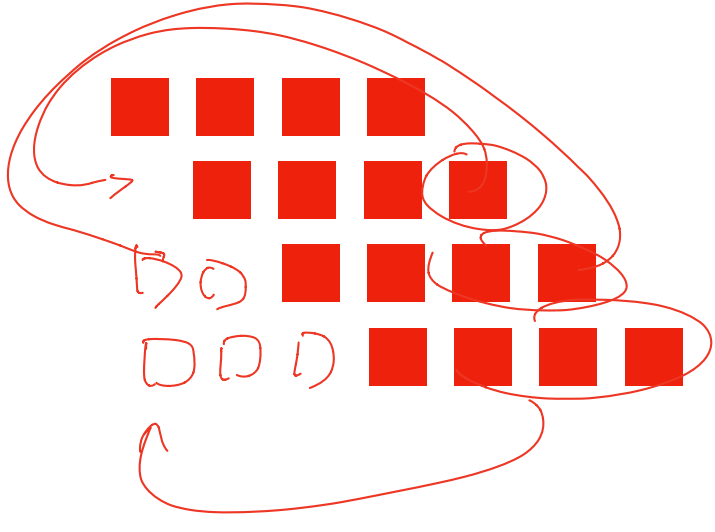


Sbox

lookup in a table.

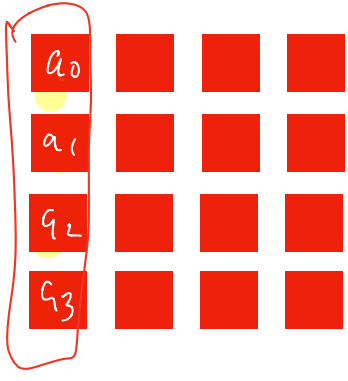
"subbytes"

AES(k,m)



shift-row

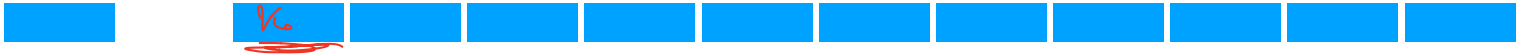
AES(k,m)

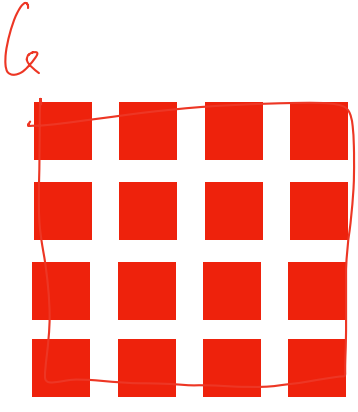


MIXCOLUMN

$$\begin{bmatrix} b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} = \begin{bmatrix} & & & \\ & A & & \\ & & & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

AES(k,m)





Add round key 1 into m

For i=1...9:

- SubBytes: apply a map to all bytes
- ShiftRows: permute the bytes
- MixColumns: permute columns
- AddRoundKey i+1

SubBytes: apply a map to all bytes

ShiftRows: permute the bytes

AddRoundKey i+1

Main security comes from s-box

AES S-Box

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value `0x9a` is converted into `0xb8`.

AES is very fast.

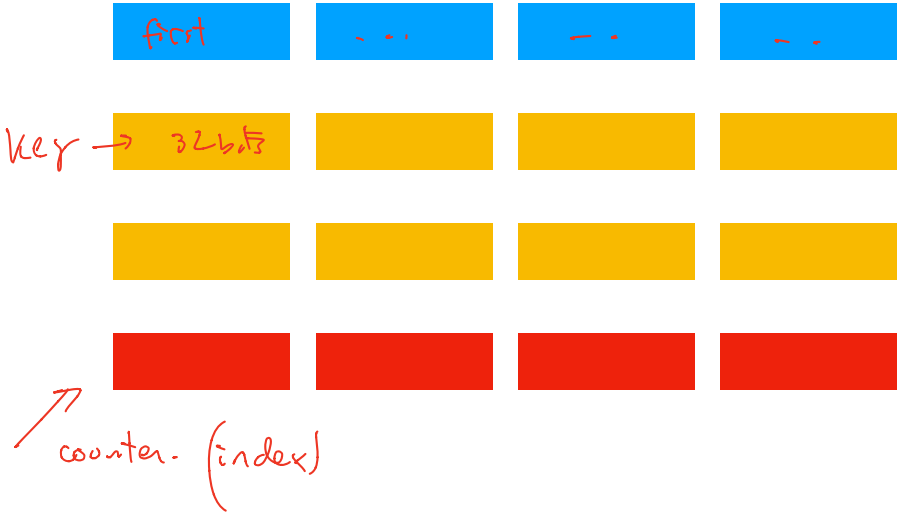
https://calomel.org/aesni_ssl_performance.html

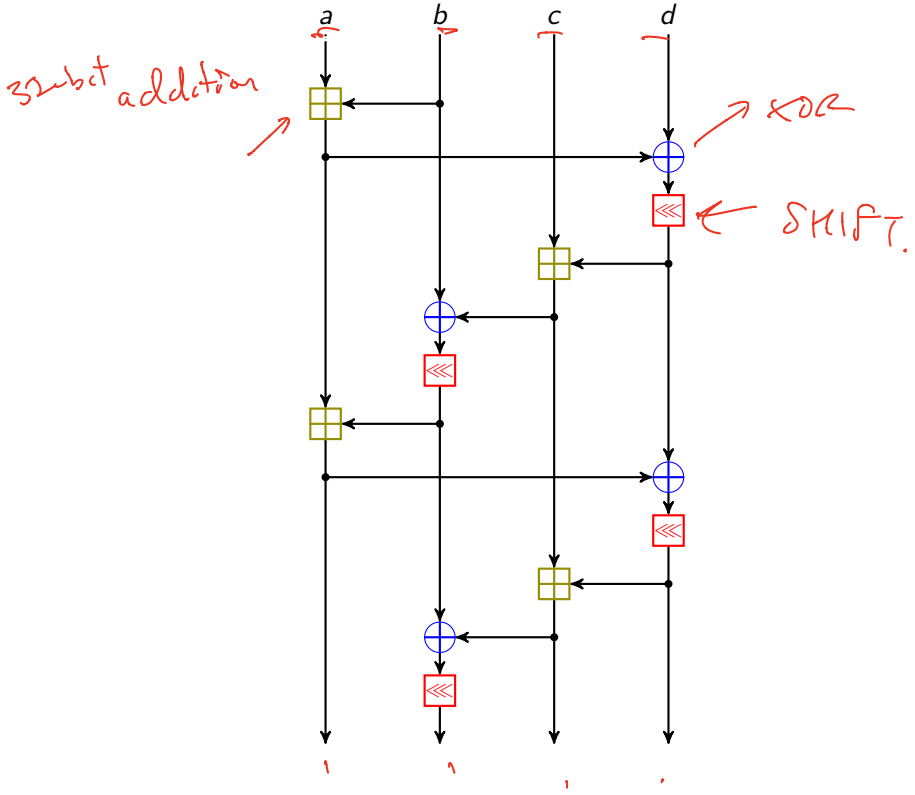
Cipher Performance per CPU core

AES Performance per CPU core for TLS v1.2 Ciphers
(Higher is Better, Speeds in Megabytes per Second)

	ChaCha20	AES-128-GCM	AES-256-GCM	AES-128-CBC	AES-256-CBC	Total Score
AMD Ryzen 7 1800X	573	3006	2642	1513	1101	= 8835
Intel W-2125	565	2808	2426	1698	1235	= 8732
Intel i7-6700	585	2607	2251	1561	1131	= 8135
AMD EPYC 7551	355	2213	1962	1114	811	= 6455
Intel i5-6500	410	1729	1520	1078	783	= 5520
Intel i7-4750HQ	369	1556	1353	688	499	= 4465
AMD FX 8350	367	1453	1278	716	514	= 4328
AMD FX 8150	347	1441	1273	716	515	= 4292
Intel E5-2650 v4	404	1479	1286	652	468	= 4289
Intel i7-2700K	382	1353	1212	763	552	= 4262
Intel i7-3840QM	373	1279	1143	725	520	= 4040
Intel i5-2500K	358	1274	1140	728	522	= 4022
AMD FX 6100	326	1344	1186	671	481	= 4008
AMD A10-7850K	321	1303	1176	685	499	= 3984
AMD A8-7600 Kaveri	306	1246	1108	648	470	= 3778
Intel E5-2640 v3	303	1286	1126	585	419	= 3719
AMD Opteron 6380	293	1203	1063	589	423	= 3571
AMD Opteron 6378	282	1138	986	561	406	= 3373
AMD Opteron 6274	232	1054	926	524	376	= 3112
Intel Xeon E5-2630	247	962	864	541	394	= 3008
Intel Xeon E5645	262	817	717	727	524	= 3047

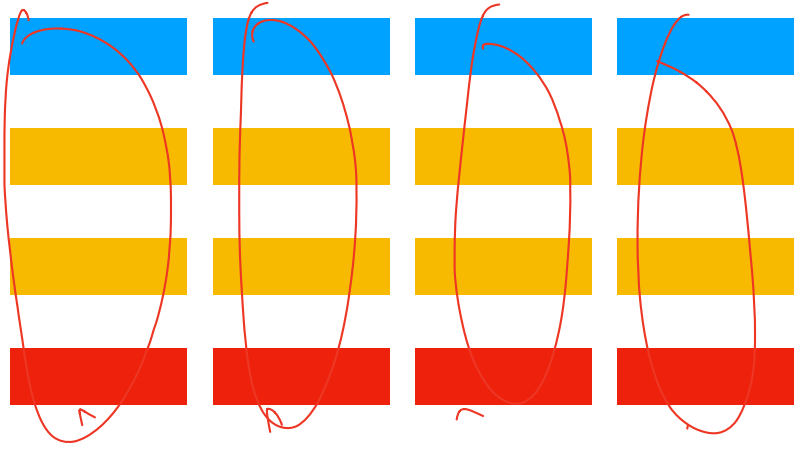
Efficiency: chacha20 (a stream cipher)




a
b
c
d
32-bit addition
XOR
SHIFT.

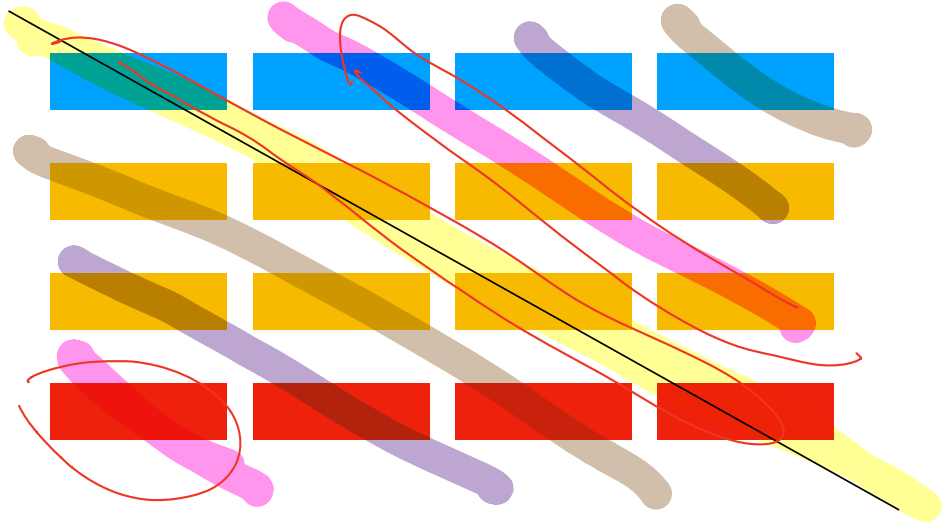
Efficiency: chacha20 (a stream cipher)

Columns



Efficiency: chacha20 (a stream cipher)

Diagonals



Chacha20

```
void chacha_block(uint32_t out[16], uint32_t const in[16])
{
  int i;
  uint32_t x[16];

  for (i = 0; i < 16; ++i)
    x[i] = in[i];
  // 10 loops × 2 rounds/loop = 20 rounds
  for (i = 0; i < ROUNDS; i += 2) {
    // Odd round
    QR(x[0], x[4], x[ 8], x[12]); // column 0
    QR(x[1], x[5], x[ 9], x[13]); // column 1
    QR(x[2], x[6], x[10], x[14]); // column 2
    QR(x[3], x[7], x[11], x[15]); // column 3
    // Even round
    QR(x[0], x[5], x[10], x[15]); // diagonal 1 (main diagonal)
    QR(x[1], x[6], x[11], x[12]); // diagonal 2
    QR(x[2], x[7], x[ 8], x[13]); // diagonal 3
    QR(x[3], x[4], x[ 9], x[14]); // diagonal 4
  }
  for (i = 0; i < 16; ++i)
    out[i] = x[i] + in[i];
}
```

Poly1305 one-time MAC

Key: (R,S)

msg: 

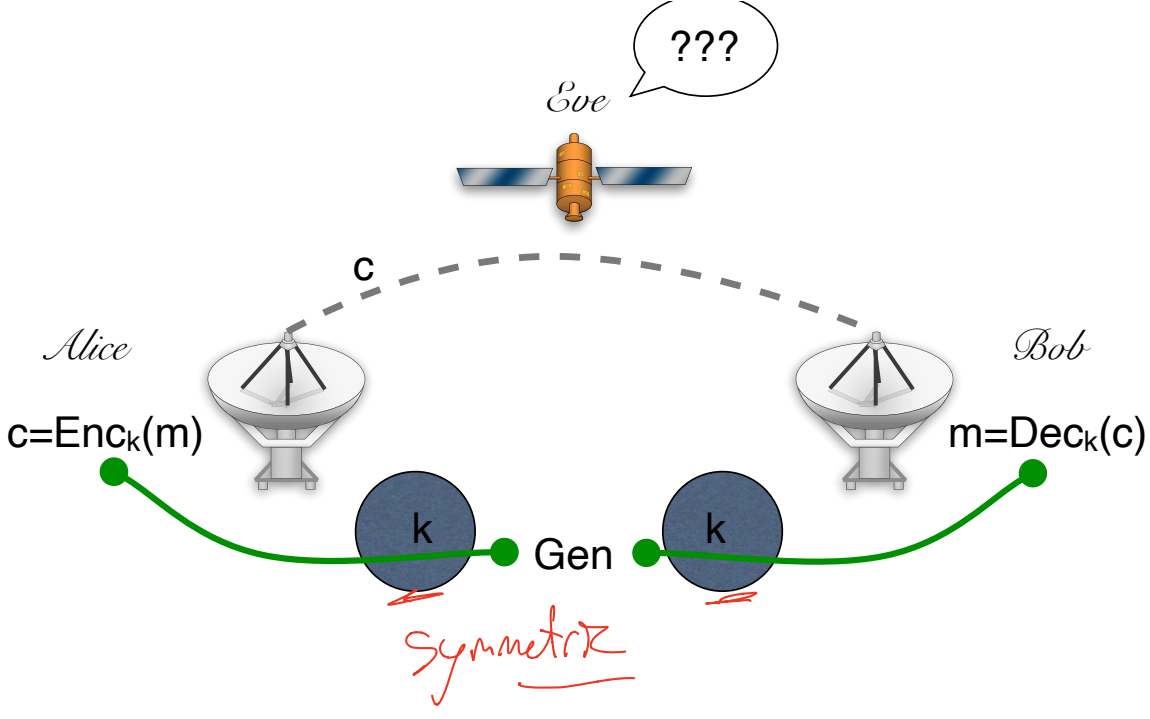
$$t = \sum_{i=0}^{n} M_{n-i} \cdot R^{n-i} \bmod (2^{130} - 5) + S \bmod 2^{128}$$

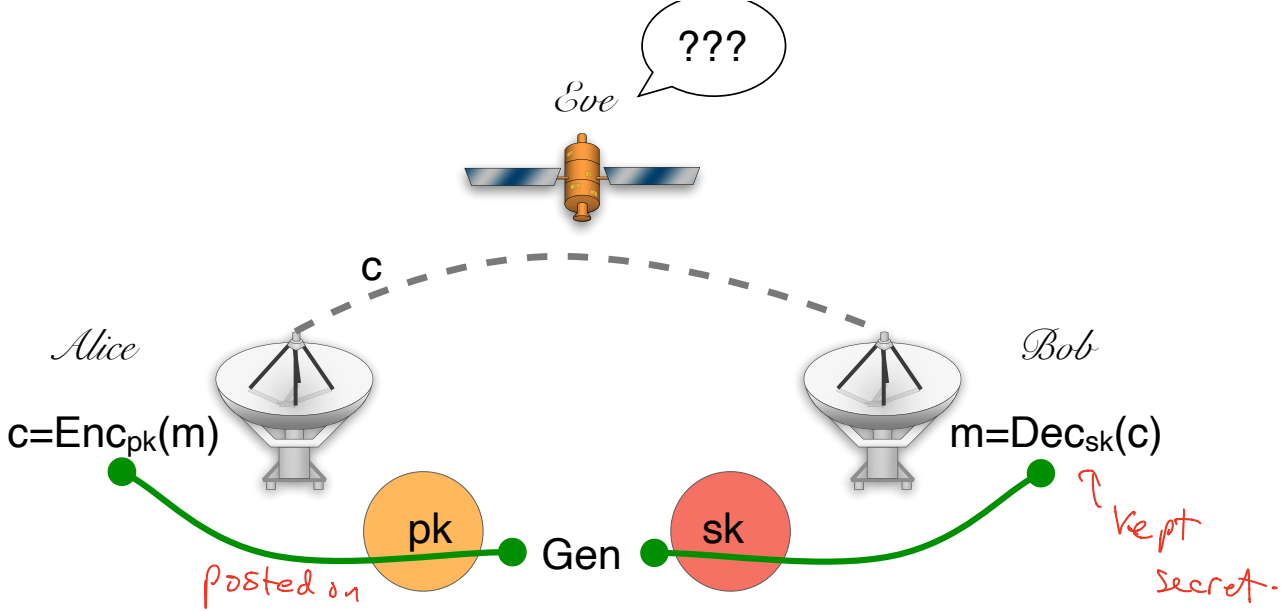
Intel Xeon E5-2690@2.9GHz with Hyper-Threading and Turbo Boost disabled

AES-128-GCM, AES-NI disabled	131 MB/s
AES-128-GCM, AES-NI enabled	892 MB/s
ChaCha20+Poly1305	427 MB/s
ChaCha20+Poly1305, -march=native	560 MB/s

https://datatracker.ietf.org/meeting/88/materials/slides-88-tls-1

Revisit our model for Encryption


???
Eve
c
Alice
Bob
c=Enc_k(m)
m=Dec_k(c)
k
Gen
k
Symmetric


???
Eve
c
Alice
Bob
c=Enc_pk(m)
m=Dec_sk(c)
pk
Gen
sk
kept secret.
posted on a bulletin board, broadcasted
asymmetric encryption
"public key model"

public key encryption

Gen Enc Dec

3 algorithms

Gen (key generation)

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$

Enc (encryption)

$c \leftarrow \mathsf{Enc}_{pk}(m)$ for $pk \in \mathcal{K}, m \in \mathcal{M}$

Dec (decryption)

$m \leftarrow \mathsf{Dec}_{sk}(c)$

public key encryption

Gen Enc Dec

3 algorithms

Gen (key generation)

$(pk, sk) \leftarrow \mathsf{Gen}(1^n)$

Enc (encryption)

$c \leftarrow \mathsf{Enc}_{pk}(m)$ for $pk \in \mathcal{K}, m \in \mathcal{M}$

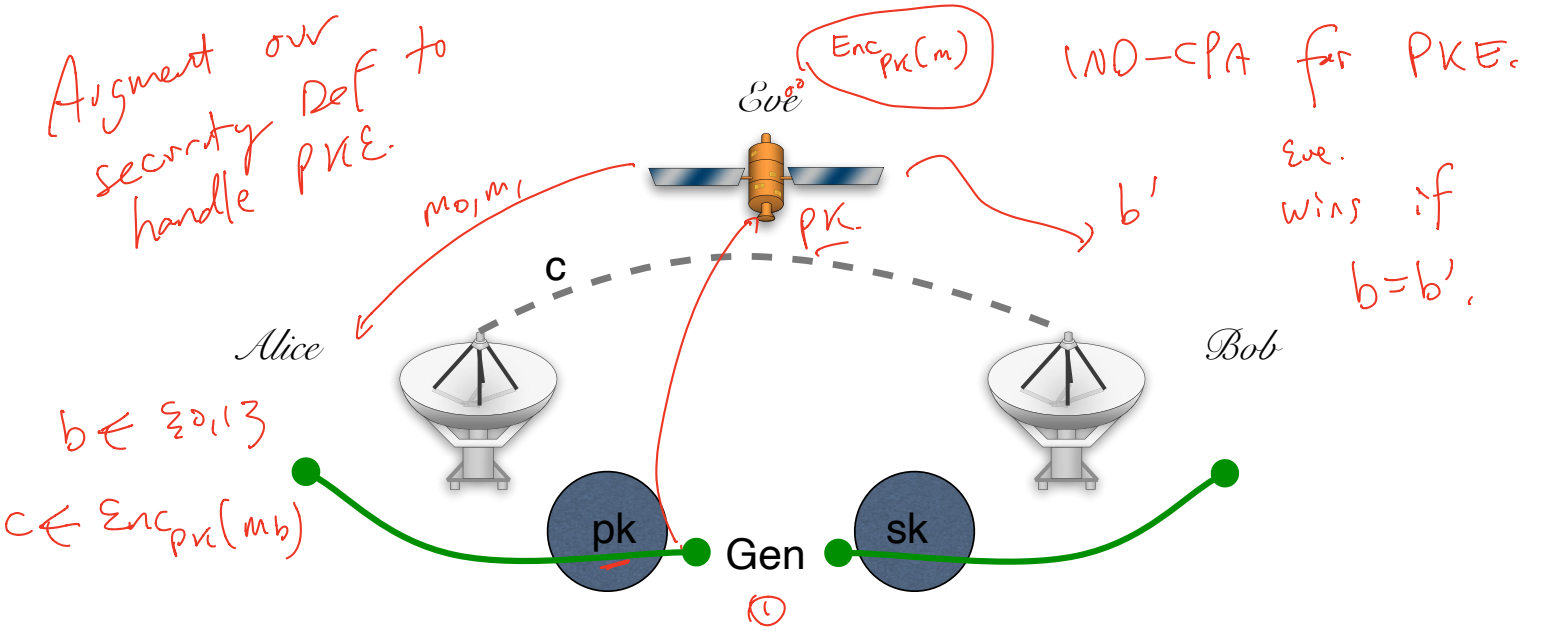
Dec (decryption)

$\forall m \in \mathcal{M}, (pk, sk) \leftarrow \mathsf{Gen}(1^n)$

$\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m] = 1$

formalize correctness

Augment our security Def to handle PKE.



IND-CPA for PKE.

Eve wins if $b = b'$.

$b \in \{0,1\}$

$c \leftarrow Enc_{pk}(m_b)$

"for any pair of messages m_1, m_2,
Eve cannot tell whether $c = Enc_{pk}(m_i)$."

IND-CPA security for pke

(weakest notion of security)

$IND\text{-}CPA(\mathcal{E}ve, 1^n)$

$(pk, sk) \leftarrow Gen(1^n)$

$(m_0, m_1) \leftarrow \mathcal{E}ve(pk, 1^n)$

$b \leftarrow \{0,1\}$

$c \leftarrow Enc_{pk}(m_b)$

$b' \leftarrow \mathcal{E}ve(c, pk, m_0, m_1)$

win if $b' = b$.

$Pr[win]$ should be negligible in n.

"i.e. 2^{-n}"

El-Gamal encryption

$\mathbf{gen}(1^n)$ — prime #, generator.

$p \leftarrow \Pi_n \quad g \leftarrow \text{Generators}_p$

$pk = g^x \bmod p \qquad sk = x.$ (DLOG)

$\mathbf{enc}_{pk}(m)$ $r \leftarrow \mathbb{Z}_p^*$ (random r in $0 \ldots p-1$)

$c_0 = g^r \bmod p. \quad c_1 = (pk)^r \cdot m \bmod p \quad (c_0, c_1)$

$\mathbf{dec}_{sk}(c)$ How can you decrypt??

$c_1 = (g^x)^r \cdot m = (g^r)^x \cdot m = (c_0)^{sk} \cdot m$

$\Rightarrow c_1 / c_0^{sk}$

El-Gamal encryption

$\mathbf{gen}(1^n)$

$p \leftarrow \Pi_n \quad g \leftarrow \text{Generators}_p$

$a \leftarrow \mathbb{Z}_p$

$pk \leftarrow (g, g^a) \quad sk \leftarrow (g, a)$

$\mathbf{enc}_{pk}(m)$

$r \leftarrow \mathbb{Z}_p$

$(g^r, (g^a)^r \cdot m)$

$\mathbf{dec}_{sk}(c)$

$(c_1, c_2) \leftarrow c$

$m \leftarrow c_2/(c_1)^a$

Example ElGamal

(skip)

Why is ElGamal secure?

decisional Diffie-Hellman assumption (DDH)

$p \leftarrow \Pi_n \quad g \leftarrow \text{Generators}_p$

$a, b, c \leftarrow \mathbb{Z}_p$ (work in a prime order group)

$$\{p, g, g^a, g^b, g^{ab}\}_n \approx \{p, g, g^a, g^b, g^c\}_n$$

under ddh, scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{Ind}_0(\Pi, \mathcal{A}, n)$

$(sk, pk) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, pk)$

$c \leftarrow \mathsf{Enc}_{pk}(m_0)$

Output $A(c, \mathrm{state})$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{Ind}_0(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_0)$

Output $A(c, \mathrm{state})$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{Ind}_0(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_0)$

Output $A(c, \mathrm{state})$

$\mathsf{H}_1(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_0)$

Output $A(c, \mathrm{state})$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{Ind}_0(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_0)$

Output $A(c, \mathrm{state})$

$\mathsf{H}_1(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_0)$

Output $A(c, \mathrm{state})$

ddh-assumption: $\{g, g^a, g^r, g^{ar}\} \approx \{g, g^a, g^r, g^z\}$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{Ind}_0(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_0)$

Output $A(c, \mathrm{state})$

$\mathsf{H}_1(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_0)$

Output $A(c, \mathrm{state})$

$$\{z \leftarrow \mathbb{Z}_q; g^z \cdot m_0\} = \{z \leftarrow \mathbb{Z}_q; g^z \cdot m_1\}$$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{H}_1(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_0)$

Output $A(c, \mathrm{state})$

$\mathsf{H}_2(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_1)$

Output $A(c, \mathrm{state})$

$$\{z \leftarrow \mathbb{Z}_q; g^z \cdot m_0\} = \{z \leftarrow \mathbb{Z}_q; g^z \cdot m_1\}$$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{H}_1(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_0)$

Output $A(c, \mathrm{state})$

$\mathsf{H}_2(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_1)$

Output $A(c, \mathrm{state})$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{H}_2(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_1)$

Output $A(c, \mathrm{state})$

$\mathsf{IND}_1(\Pi, \mathcal{A}, n)$

$((g, a), (g, g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_1)$

Output $A(c, \mathrm{state})$

under DDH,
scheme is cpa-secure

$\mathsf{enc}_{\mathsf{pk}}(\mathsf{m})$

$r \leftarrow \mathbb{Z}_q$

$(g^r, (g^a)^r \cdot m)$

$\mathsf{H}_2(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^z \cdot m_1)$

Output $A(c, \mathrm{state})$

$\mathsf{IND}_1(\Pi, \mathcal{A}, n)$

$((g,a),(g,g^a)) \leftarrow \mathrm{Gen}(1^n)$

$m_0, m_1, \mathrm{state} \leftarrow A(1^n, (g, g^a))$

$c \leftarrow (g^r, g^{ar} \cdot m_1)$

Output $A(c, \mathrm{state})$

ddh-assumption: $\{g, g^a, g^r, g^{ar}\} \approx \{g, g^a, g^r, g^z\}$

GPG RSA (PKCS v1.5)

$\text{gen}(1^n)$

$SK \leftarrow p, q, d$ (2 randomly selected primes)

$PK = (N, e)$ → 65537 (prime)

$\text{enc}_{pk}(m)$

$\text{dec}_{sk}(c)$

GPG RSA (PKCS v1.5)

gen(1^n)

$$N \leftarrow pq, p, q \in \Pi_n, e \in \mathbb{Z}^*_{\phi(n)}$$

$$pk \leftarrow (N, e) \qquad sk \leftarrow (N, d)$$

$\text{enc}_{pk}(m)$

$\text{dec}_{sk}(c)$

GPG RSA (PKCS v1.5)

gen(1^n)

$$N \leftarrow pq, p, q \in \Pi_n, e \in \mathbb{Z}^*_{\phi(n)}$$

$$pk \leftarrow (N, e) \qquad sk \leftarrow (N, d)$$

enc_{pk}(m)

$$c \leftarrow m^e \bmod N$$

dec_{sk}(c)

$$m \leftarrow c^d \bmod N$$

Textbook RSA (broken)

GPG RSA (PKCS v1.5)

gen(1^n)

$$N \leftarrow pq, p, q \in \Pi_n, e \in \mathbb{Z}^*_{\phi(n)}$$

$$pk \leftarrow (N, e) \qquad sk \leftarrow (N, d)$$

enc_{pk}(m)

$$c \leftarrow m^e \bmod N$$

dec_{sk}(c)

$$m \leftarrow c^d \bmod N$$

pkcs1.5

$\text{enc}_{pk}(m)$

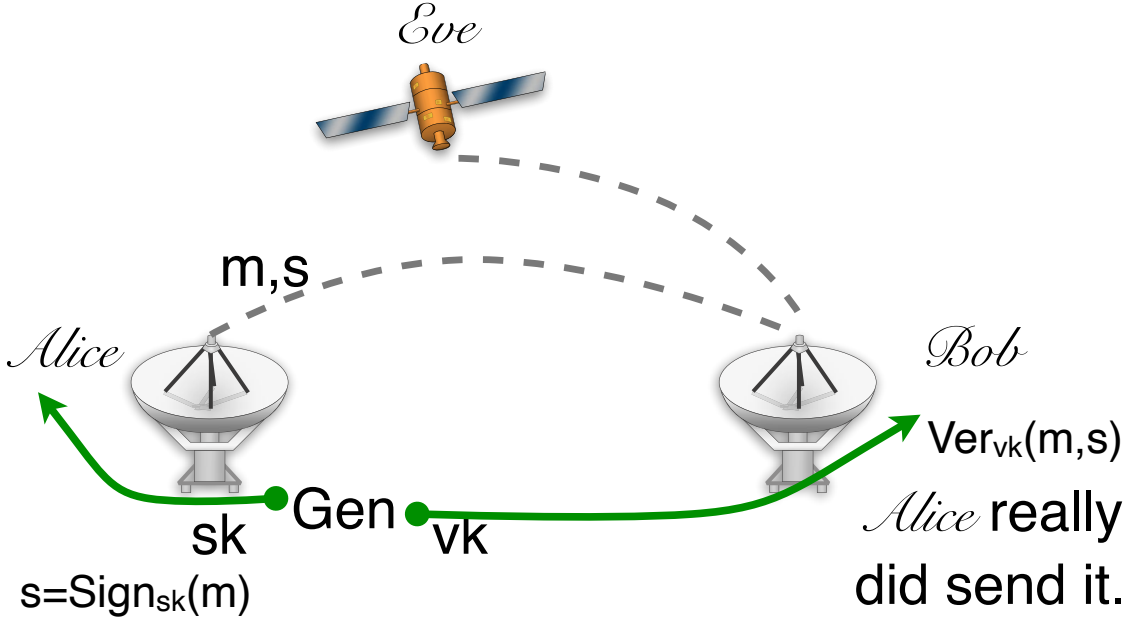
pick r as a random string with no 0s

(typically 8 bytes)

$$c \leftarrow (0||2||r||0||m)^e \bmod N$$

"padding oracle" attack against this scheme

Public key digital signature



Public key digital signature

message space $\{\mathcal{M}\}_n$

$\mathsf{Gen}(1^n)$

$\mathsf{Sign}_{sk}(m)$

$\mathsf{Ver}_{vk}(m,s)$

Public key digital signature

message space $\{\mathcal{M}\}_n$

$\mathrm{Gen}(1^n)$ generates a key pair sk,vk

$\mathrm{Sign}_{sk}(m)$

$\mathrm{Ver}_{vk}(m,s)$

Public key digital signature

message space $\{\mathcal{M}\}_n$

Gen(1^n) generates a key pair sk,vk

$\text{Sign}_{sk}(m)$ generates a signature s for $m \in \mathcal{M}_n$

$\text{Ver}_{vk}(m,s)$

Public key digital signature

message space $\{\mathcal{M}\}_n$

Gen(1^n) generates a key pair sk,vk

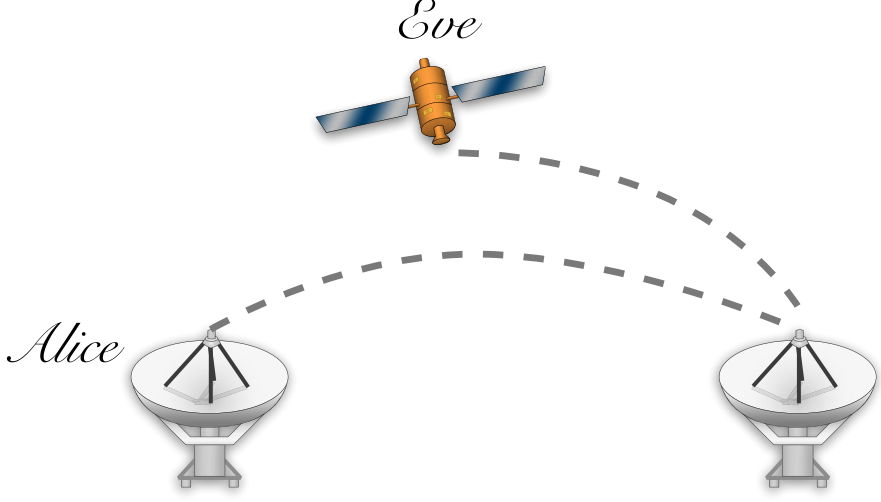
$\text{Sign}_{sk}(m)$ generates a signature s for $m \in \mathcal{M}_n$

$\text{Ver}_{vk}(m,s)$ accepts or rejects a msg,sig pair

$$\Pr[k \leftarrow Gen(1^n) : Ver_{vk}(m, Sign_{sk}(m)) = 1] = 1$$

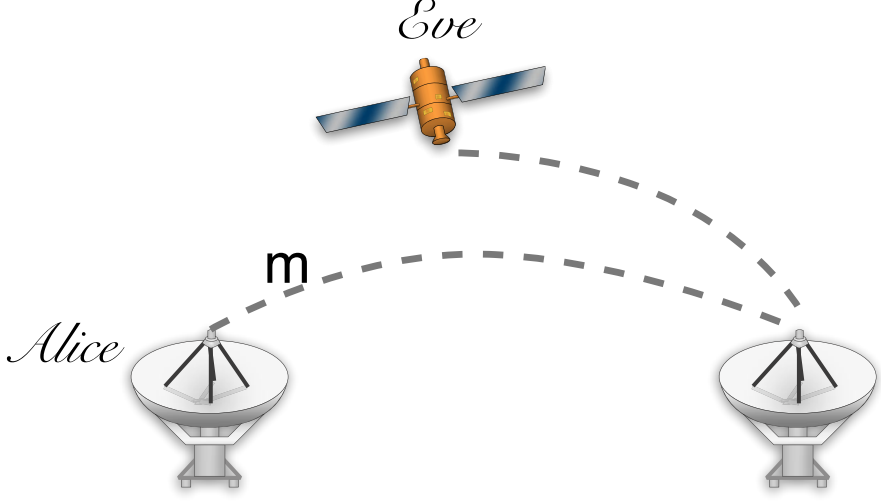
existential unforgability

"even when given a signing oracle,
an adversary cannot forge a signature for
any message of its choosing "



existential unforgability

"even when given a signing oracle,
an adversary cannot forge a signature for
any message of its choosing "



for all non-uniform ppt A

$$\Pr\Big[\qquad\qquad\Big] < \mu(n)$$

for all non-uniform ppt A

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow Gen(1^n); (m, s) \leftarrow A^{Sign_{sk}(\cdot)} : \\ \quad Ver_{vk}(m, s) = 1 \\ \quad \text{and A didn't query m}\end{array}\right] < \mu(n)$$

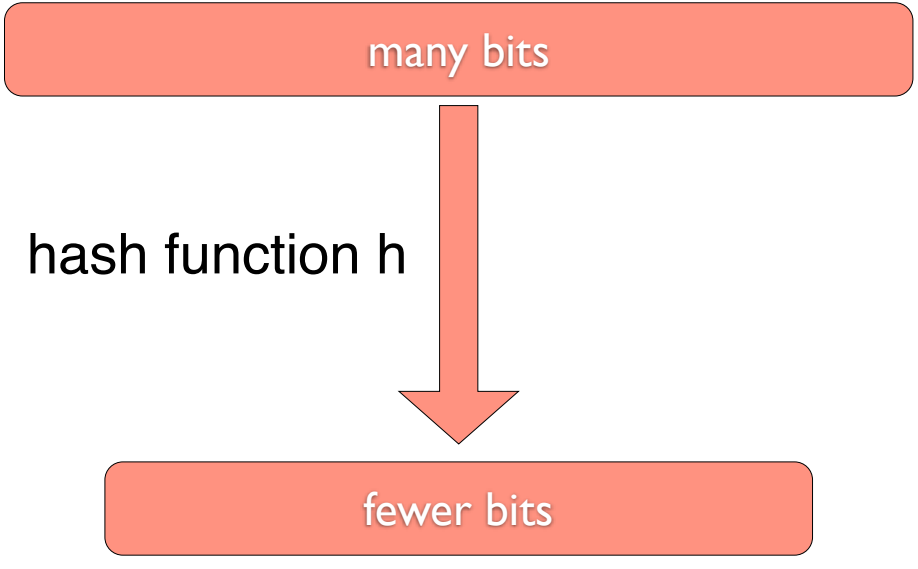
RSA Signatures in GPG

Sign((sk, N) m):

Compute the padding: $z \leftarrow 00 \cdot 01 \cdot FF \cdots FF \cdot 00 \cdot \mathsf{ID}_H \cdot H(m)$

Compute the signature: $\sigma \leftarrow z^{sk} \bmod N$

goal of a hash function



a hash function is a function

$$h : \{0,1\}^d \rightarrow \{0,1\}^r$$

such that h is easy to evaluate
and r < d

useful in data structures

```
public class test
{
    public static void main(String[] args)
    {
        System.out.println(args[0].hashCode());
    }
}
```

```
abhi$ java test HHHHHHHHHHHHHHHHHHHHHGGGDD
-1644493785
```

collisions should be rare

```
public class test
{
    public static void main(String[] args)
    {
        System.out.println(args[0].hashCode());
    }
}
```

```
abhi$ java test HHHHHHHHHHHHHHHHHHHHHGGGDD
-1644493785

abhi$ java test “hello world”
1794106052
```

java hash function

$$h(s) = \sum_{i=0}^{n} s[i]31^{n-i}$$

java hash function

$$h(s) = \sum_{i=0}^{n} s[i]31^{n-i}$$

it is thus easy to find a pair s_1, s_2 such that $h(s_1) = h(s_2)$

```
public class test
{
    public static void main(String[] args)
    {
        System.out.println(args[0].hashCode());
    }
}
```

```
abhi$ java test HHHHHHHHHHHHHHHHHHHHHGGGDD
-1644493785
```

```
public class test
{
    public static void main(String[] args)
    {
        System.out.println(args[0].hashCode());
    }
}
```

```
abhi$ java test HHHHHHHHHHHHHHHHHHHHHGGGDD
-1644493785

abhi$ java test HHHHHHHHHHHHHHHHHHHHHGGGCc
-1644493785
```

```
public class test
{
    public static void main(String[] args)
    {
        System.out.println(args[0].hashCode());
    }
}
```

```
abhi$ java test HHHHHHHHHHHHHHHHHHHHGGGDD
-1644493785

abhi$ java test HHHHHHHHHHHHHHHHHHHHGGGCc
-1644493785
```

‘D’ - ‘c’ + 31(‘D’-‘C’) = 0

Collision resistant hash function

in addition to being easy to compute,
it should be “hard” for a p.p.t. adversary
to find a hash collision.

md4	1990
md5	1992
sha1	1994
sha256	2005
Sha3	2015

md4	1990	128 bit
md5	1992	128 bit
sha1	1994	160 bit
sha256	2005	256 bit
Sha3	2015	

md4	1990	128 bit	1995
md5	1992	128 bit	1998
sha1	1994	160 bit	2005*
sha256	2005	256 bit	
Sha3	2015		

Recap: