Implementation Security In Cryptography

Lecture 03: Introducing Block Ciphers

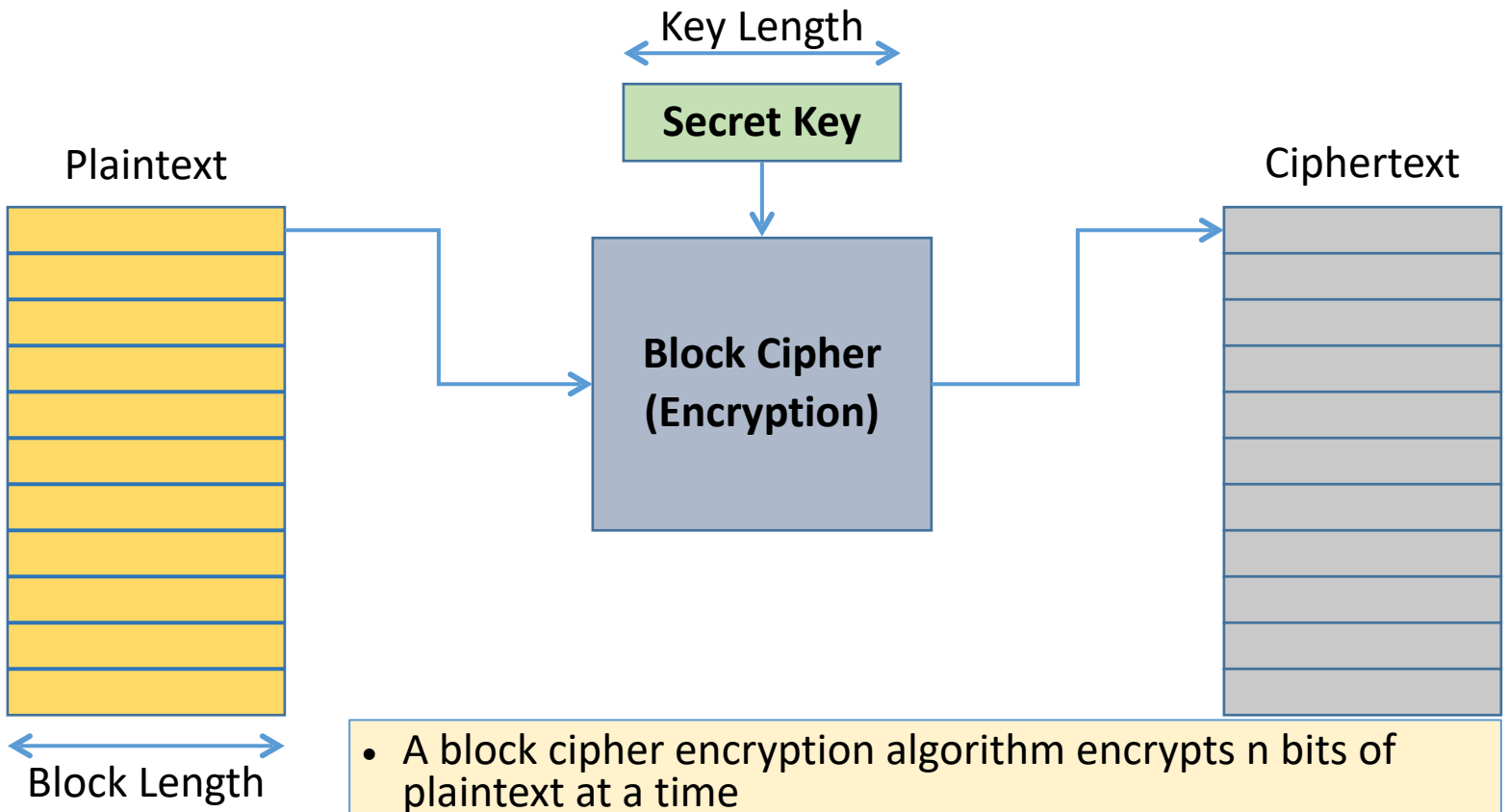
Recap

- In the last lecture
  - What people expect from crypto?
  - How they theoretically model the adversaries and the security?
  - PRG and PRFs — important building blocks for symmetric key crypto
- Public key also have similar set of security definitions with some more building blocks based on “hard problems”
- We can always “prove” if these building blocks are secure, then the crypto system is secure..
- But for this course, we are more interested in
  - How these core building blocks are implemented.
  - Are they really secure in the practical world??

Today..

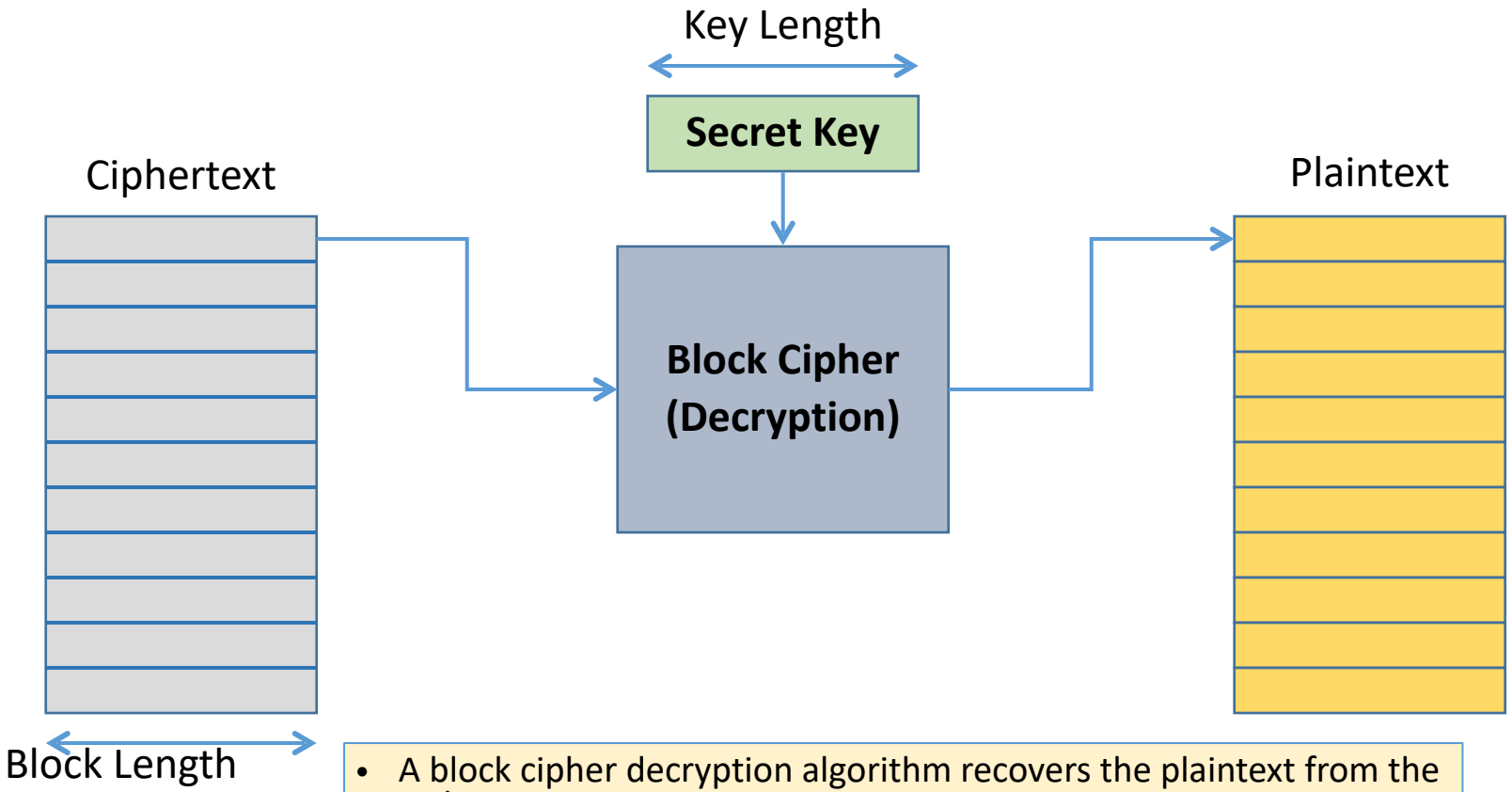
- Block ciphers — building blocks for PRFs, and sometimes PRG too..
  - PRESENT
  - AES

Block Cipher : A Symmetric Key Encryption



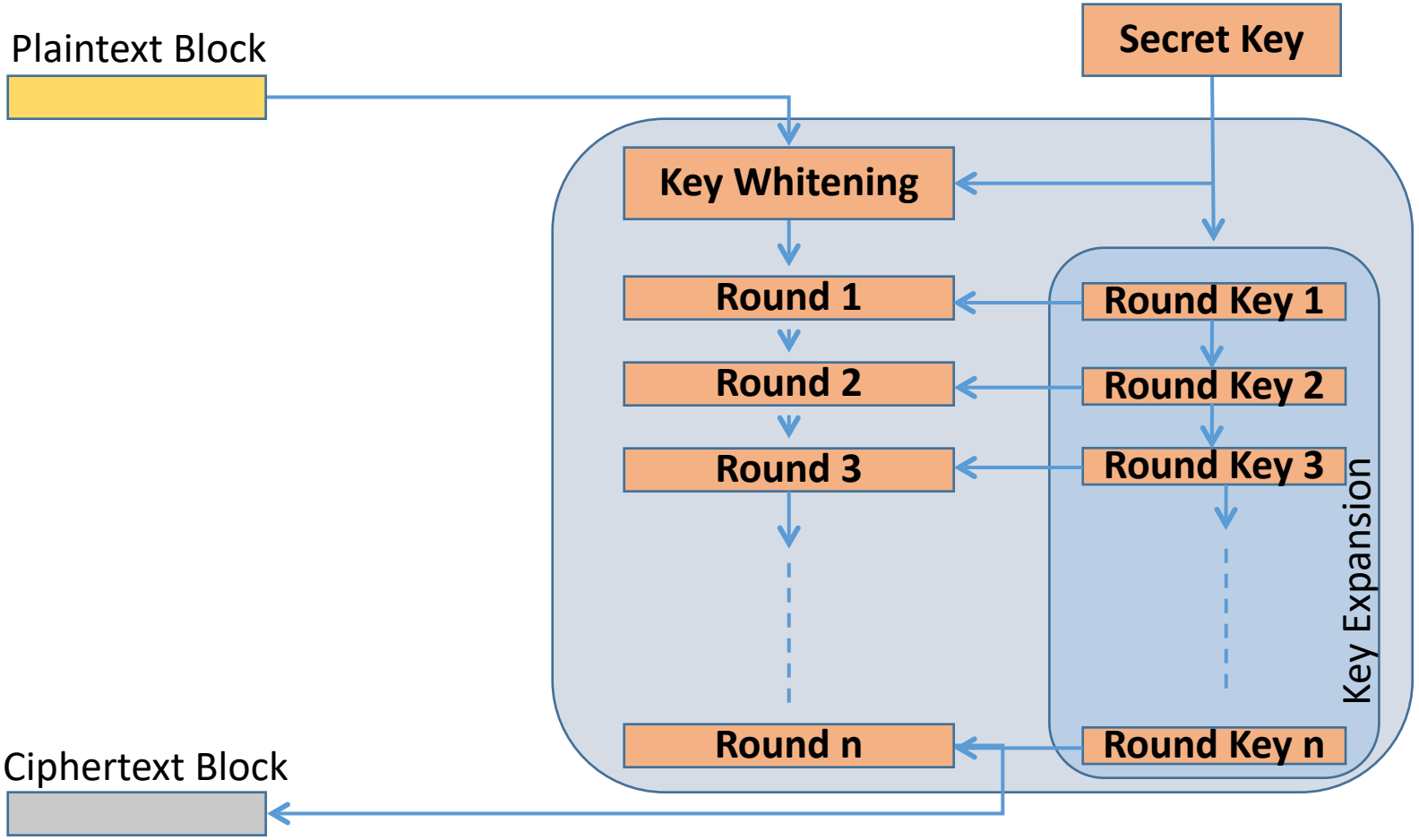
- A block cipher encryption algorithm encrypts n bits of plaintext at a time
- $C = E_k(P)$

Block Cipher : Decryption



- A block cipher decryption algorithm recovers the plaintext from the ciphertext.
- $P = D_k(C)$

Structure of a Block Cipher

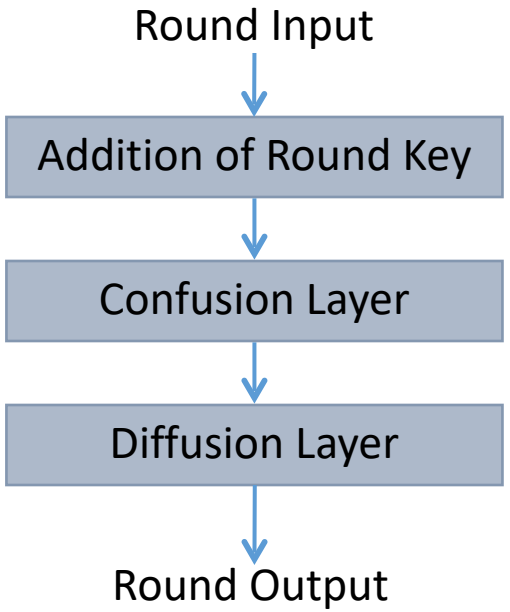


Key and Block length

- Generally 128, 192, or 256 bits
- The key and block length can be independent of each other
- The choice of length decides how effective the cipher is against brute force attacks.
  - 80 bits is considered the minimum requirement.
- Longer lengths however result in slower encryption times, thus more performance overheads.

An Encryption Round

- Consists of three operations
  - Addition of Round Key
  - ***Confusion :*** Hides the relation between the ciphertext and the key.
  - ***Diffusion :*** Hides the relation between the ciphertext and the plaintext.
- Number of rounds is determined by the cipher's resistance to known attacks.

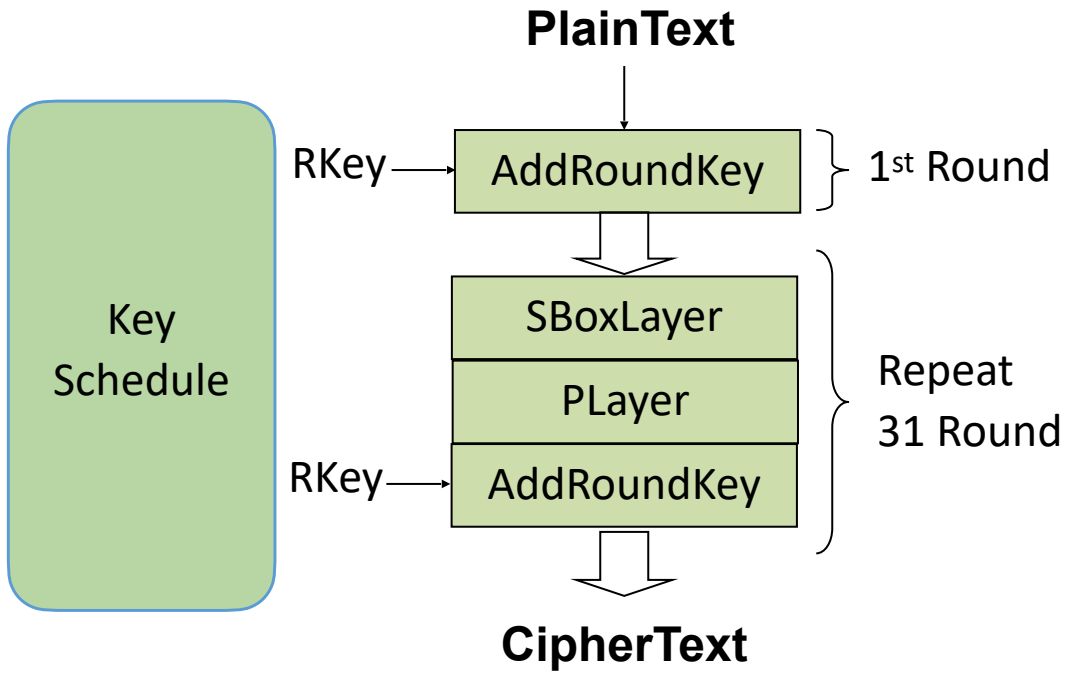


PRESENT: A Lightweight Block Cipher

- Made for lightweight applications.
- ISO standard for lightweight cryptography
- 64-bit block size
- 80/128 bit key size
- 31 rounds
- Now let us see how it is made

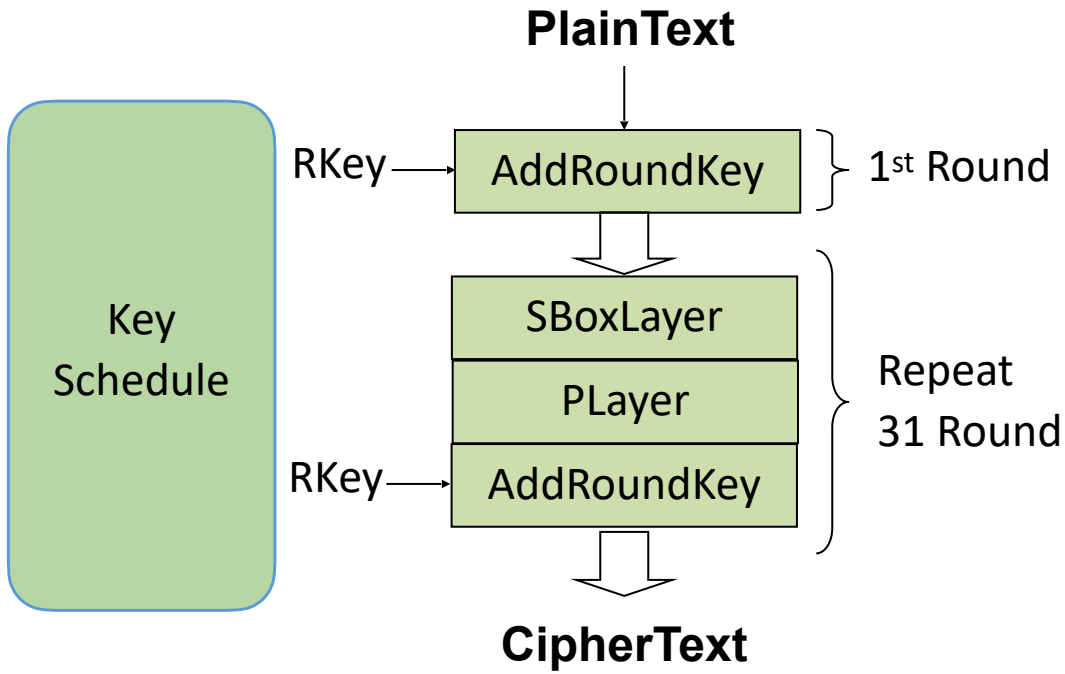
PRESENT: A Lightweight Block Cipher

- Made for lightweight applications.
- ISO standard for lightweight cryptography
- 64-bit block size
- 80/128 bit key size
- 64-bit Round keys
- Now let us see how it is made…



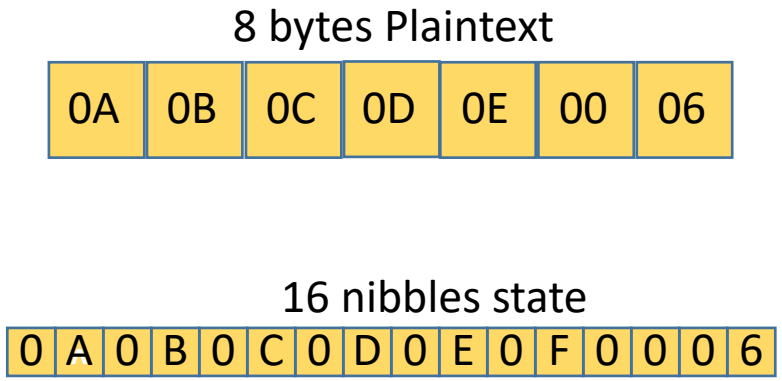
PRESENT: A Lightweight Block Cipher

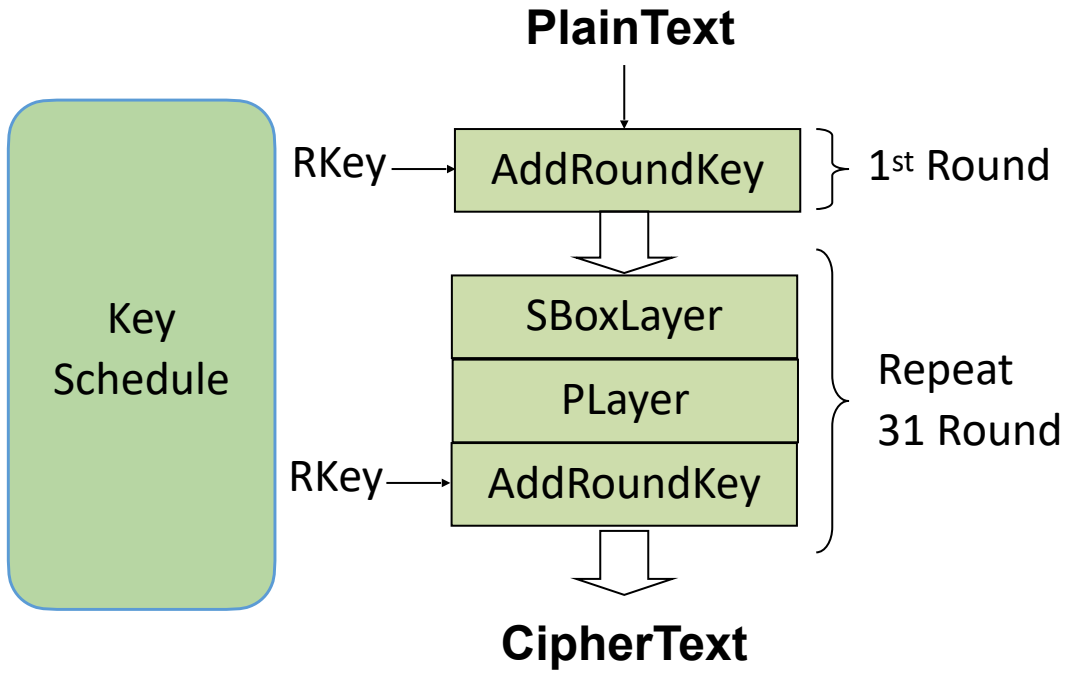
- Everything is defined as Boolean logic
- For convenience we shall use a special kind of logic representation style
  - Algebraic Normal Form (ANF)
  - The functionally complete set is (AND, XOR, 0, 1)
  - Also, a bit of notation abuse: + is $\oplus$
- Let's see a bit of ANF
  - How do we represent NOT: a + 1
  - How do we present OR: a + b + ab — see why?



PRESENT: A Lightweight Block Cipher

- The State Representation

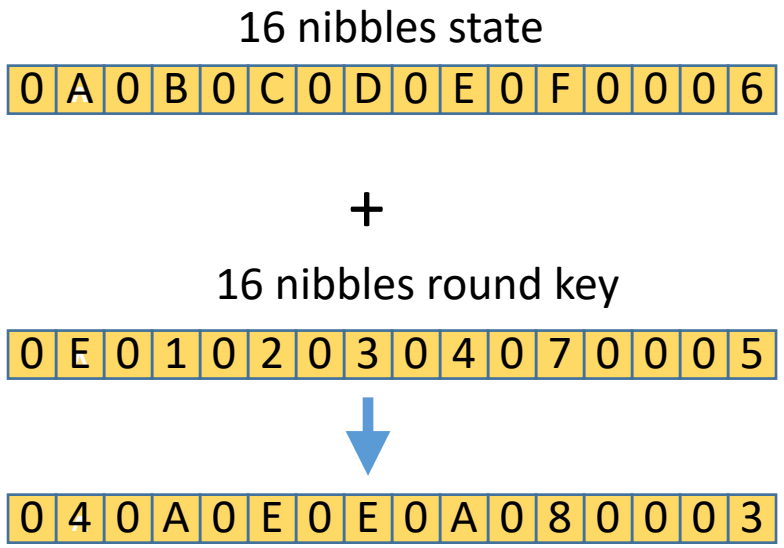


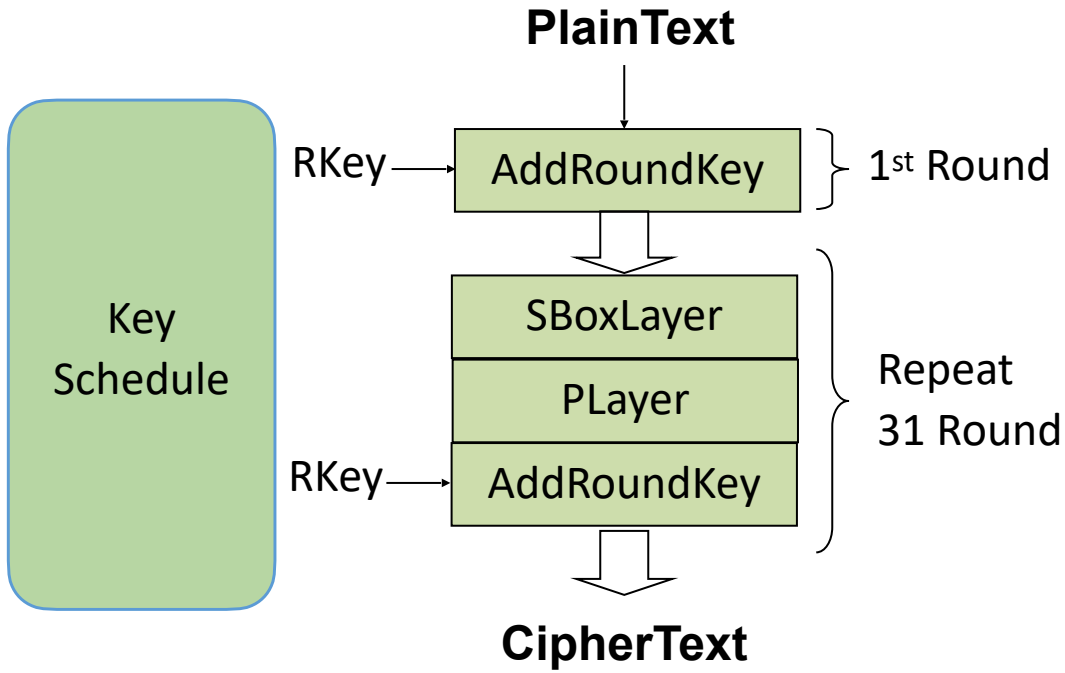


- Nibble: 4-bit chunk of the state.
- In hardware — straightforward
- In software — each 32 bit register contains 8 nibbles. So the entire state can be contained in 2 registers (embedded world).

PRESENT: A Lightweight Block Cipher

- AddRoundKey

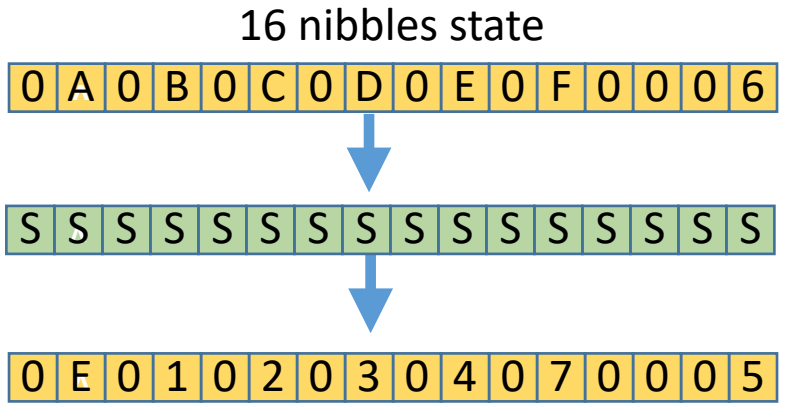


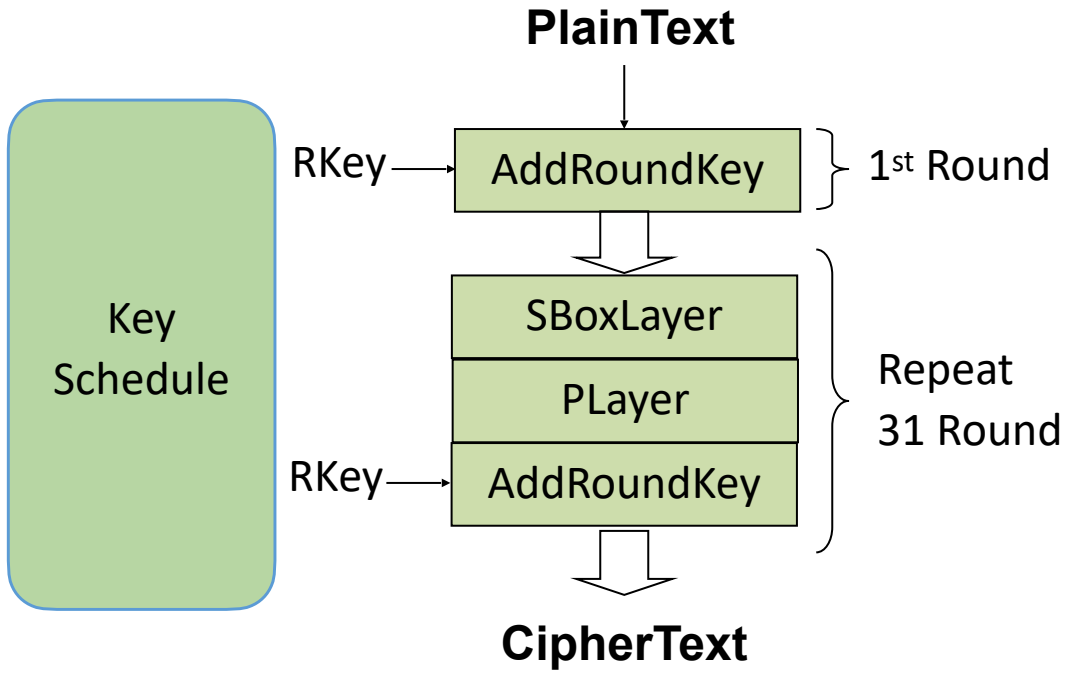


- Remember "+" means XOR
- Bitwise XOR operation
- Very easy to do in both software and hardware

PRESENT: A Lightweight Block Cipher

- SBoxLayer





x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

- S-Box is a bijection from 4-bits to 4-bits
- It is not “just” some random mapping
- It has certain properties which makes it resistant against differential and other attacks

PRESENT: A Lightweight Block Cipher

- SBoxLayer

- It has certain properties which makes it resistant against differential and other attacks
  - Firstly, it is a balanced function
  - Secondly, one bit change in input at least propagate to 2 output bits of the S-Box
  - There are many more to prevent against differential and linear attacks

$$y_1 = x_1x_2x_4 + x_1x_3x_4 + x_1 + x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1$$

$$y_2 = x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + x_1 + x_2 + x_3x_4 + 1$$

$$y_3 = x_1x_2x_4 + x_1x_2 + x_1x_3x_4 + x_1x_3 + x_1 + x_2x_3x_4 + x_3$$

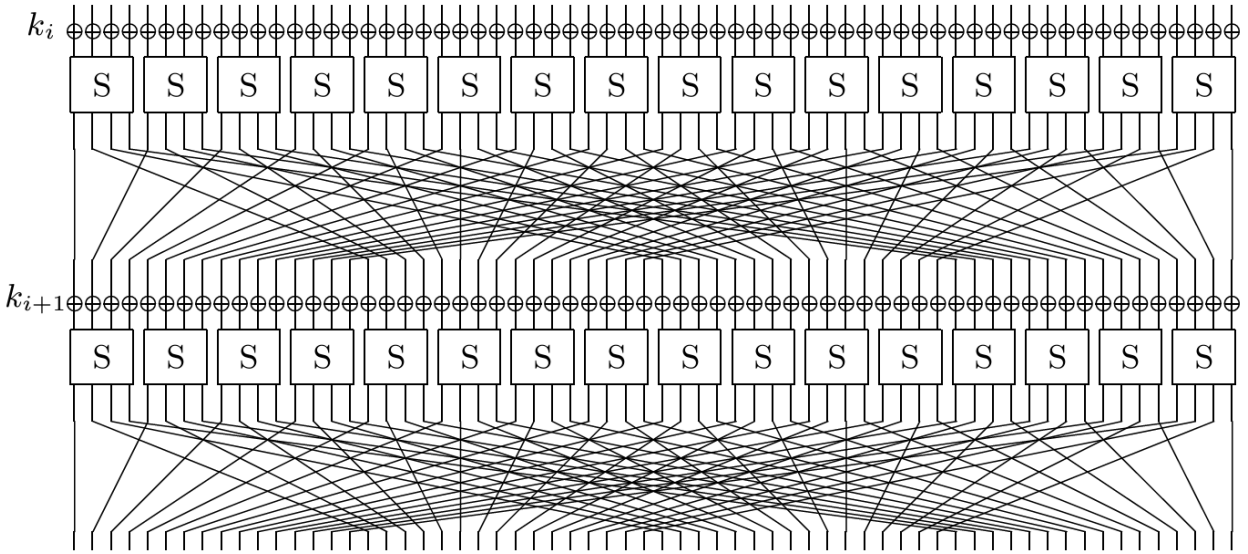
$$y_4 = x_1 + x_2x_3 + x_2 + x_4$$

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

PRESENT: A Lightweight Block Cipher

- PLayer

- It is basically a bit-permutation
  - But it has also got certain properties
  - Especially, it should ensure high number of **active S-Boxes**
  - **No cost in hardware…But tricky to implement in software**

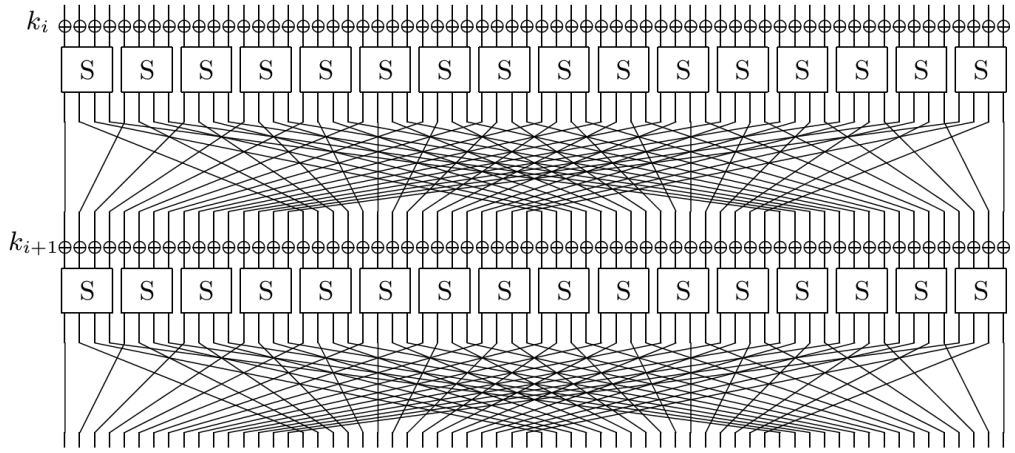


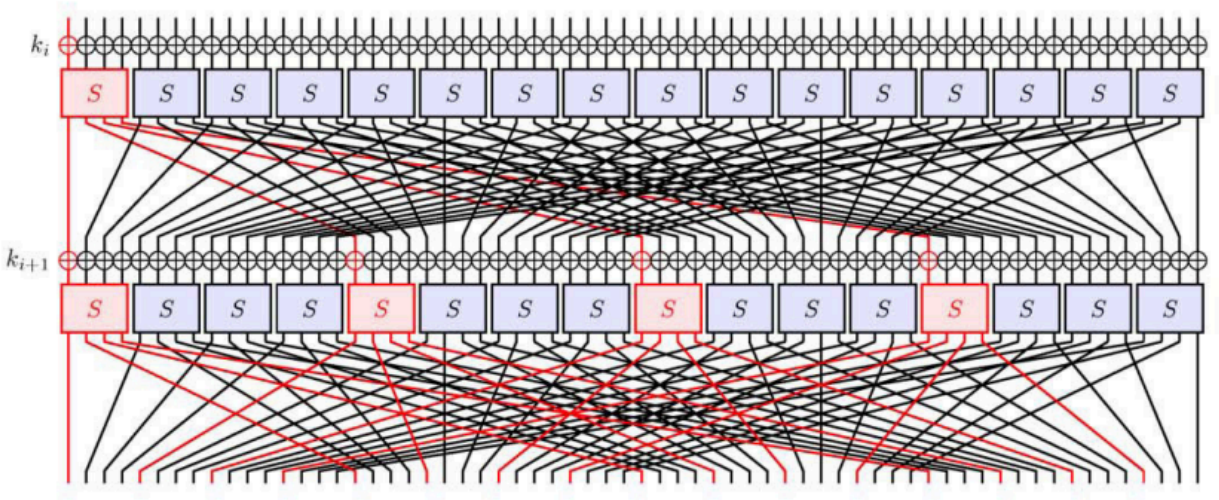
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

PRESENT: A Lightweight Block Cipher

- Key Schedule

- It is basically a bit-permutation
  - But it has also got certain properties
  - Especially, it should ensure high number of **active S-Boxes**
  - **No cost in hardware…But tricky to implement in software**





i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

PRESENT: A Lightweight Block Cipher

- Key Schedule

- Let us denote the 80-bit master key as $K = k_{79}k_{78}\cdots k_0$
- Round key: $K_i = \kappa_{63}\kappa_{62}\cdots\kappa_0 = k_{79}k_{78}\cdots k_{16}$
- Key update:

1. $[\mathtt{k}_{79}\mathtt{k}_{78}\ldots\mathtt{k}_1\mathtt{k}_0] = [\mathtt{k}_{18}\mathtt{k}_{17}\ldots\mathtt{k}_{20}\mathtt{k}_{19}]$
2. $[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}] = S[\mathtt{k}_{79}\mathtt{k}_{78}\mathtt{k}_{77}\mathtt{k}_{76}]$
3. $[\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] = [\mathtt{k}_{19}\mathtt{k}_{18}\mathtt{k}_{17}\mathtt{k}_{16}\mathtt{k}_{15}] \oplus \mathtt{round\_counter}$

PRESENT: A Lightweight Block Cipher

- How to Decrypt

- Just perform the sequence of operations in reverse.
- Need the inverse of the S-Box
- Need the inverse permutation
- Can you calculate them? — try it in Python or Sage

Examples from Public Key World

- Next few slides are adopted from:
- https://www.cs.purdue.edu/homes/ninghui/courses/426_Fall10/handouts/426_Fall10_lect31.ppt

RSA Algorithm

- Invented in **1978** by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
  - Published as R L Rivest, A Shamir, L Adleman, "*On Digital Signatures and Public Key Cryptosystems*", Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978
- Security relies on the difficulty of factoring large composite numbers

RSA Public Key Crypto System

Key generation:

1. Select 2 large prime numbers of about the same size, p and q
   Typically each p, q has between 512 and 2048 bits
2. Compute n = pq, and $\Phi(n) = (q-1)(p-1)$
3. Select e, $1<e<\Phi(n)$, s.t. $\gcd(e, \Phi(n)) = 1$

4. Compute d, $1< d<\Phi(n)$ s.t. $ed \equiv 1 \bmod \Phi(n)$
   Knowing $\Phi(n)$, d easy to compute.

Public key: (e, n)

Private key: d

RSA Description (cont.)

Encryption

Given a message M, $0 < M < n$ $\quad M \in Z_n - \{0\}$

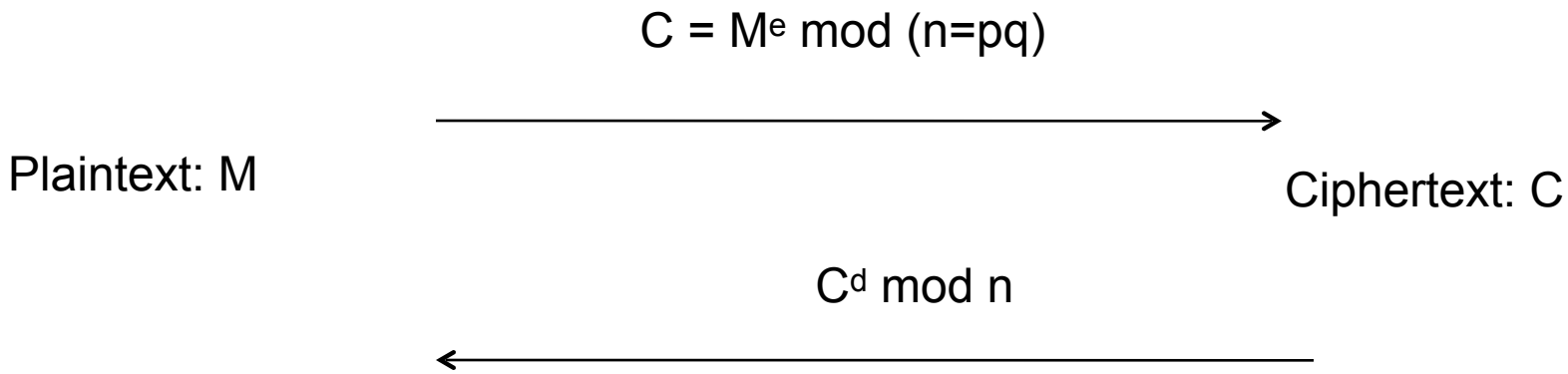
use public key (e, n)

compute $C = M^e \bmod n$ $\quad C \in Z_n - \{0\}$

Decryption

Given a ciphertext C, use private key (d)

Compute $C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n = M$



From n, difficult to figure out p,q

From (n,e), difficult to figure d.

From (n,e) and C, difficult to figure out M s.t. $C = M^e$

RSA Example

- p = 11, q = 7, n = 77, $\Phi(n) = 60$
- d = 13, e = 37 (ed = 481; ed mod 60 = 1)
- Let M = 15. Then $C \equiv M^e \bmod n$
  - $C \equiv 15^{37} \pmod{77} = 71$
- $M \equiv C^d \bmod n$
  - $M \equiv 71^{13} \pmod{77} = 15$

RSA Example 2

- Parameters:
  - p = 3, q = 5, q= pq = 15
  - Φ(n) = ?
- Let e = 3, what is d?
- Given M=2, what is C?
- How to decrypt?

RSA Security

- Security depends on the difficulty of factoring n
  - Factor n => $\Phi(n)$ => compute d from (e, $\Phi(n)$)
- The length of n=pq reflects the strength
  - 700-bit n factored in 2007
  - 768 bit factored in 2009
- 1024 bit for minimal level of security today
  - likely to be breakable in near future
- Minimal 2048 bits recommended for current usage
- NIST suggests 15360-bit RSA keys are equivalent in strength to 256-bit
- RSA speed is quadratic in key length

Real World Usage of Public Key Encryption

- Often used to encrypt a symmetric key
  - To encrypt a message M under a public key (n,e), generate a new AES key K, compute [RSA(n,e,K), AES(K,M)]
- Plain RSA does not satisfy IND requirement.
  - How to break it? — think about it
- One often needs padding, e.g., Optimal Asymmetric Encryption Padding (OAEP)
  - Roughly, to encrypt M, chooses random r, encode M as
    $M' = [X = M \oplus H_1(r)\ ,\ Y = r \oplus H_2(X)]$ where
    H_1 and H_2 are cryptographic hash functions, then encrypt it as $(M')^e \bmod n$
  - Note that given M'=[X,Y], $r = Y \oplus H_2(X)$, and $M = X \oplus H_1(r)$

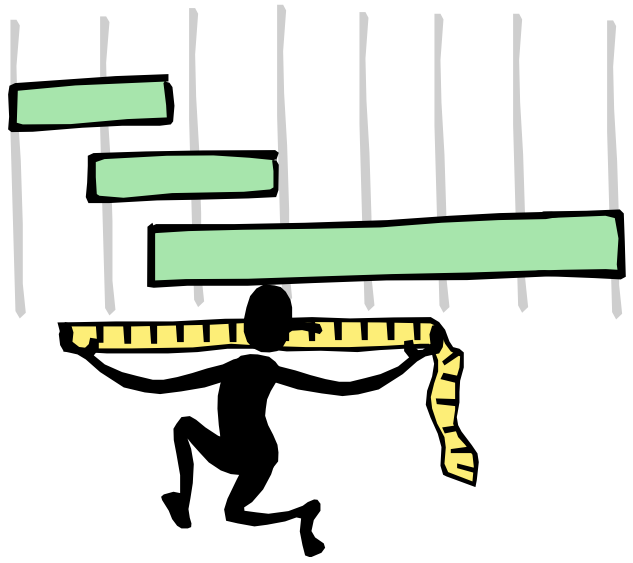
The Big Picture

	Secret Key Setting	Public Key Setting
Secrecy / Confidentiality	Stream ciphers Block ciphers + encryption modes	Public key encryption: RSA, El Gamal, etc.
Authenticity / Integrity	Message Authentication Code	Digital Signatures: RSA, DSA, etc.

Hash and MAC

- Very often digital signatures are used with hash functions, hash of a message is signed, instead of the message.
- **Hash function** creates a small fixed length digest from arbitrary length message
  - $H : \{0,1\}^* \longrightarrow \{0,1\}^n$
- Hash function must be:
  - Pre-image resistant
  - Weak collision resistant
  - Strong collision resistant

– **Message Authentication Code (MAC):**
  – **Integrity and authentication using symmetric key only.**
  – **Hash then use a block cipher to encrypt the hash**
  – **The other party must have the secret to verify.**



Digital Signatures: The Problem

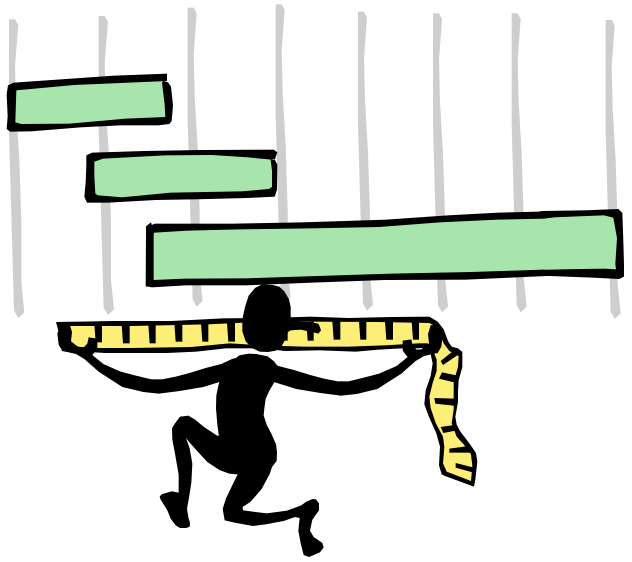
- Consider the real-life example where a person pays by credit card and signs a bill; the seller verifies that the signature on the bill is the same with the signature on the card
- Contracts, they are valid if they are signed.
- Signatures provide non-repudiation.
  - ensuring that a party in a dispute cannot repudiate, or refute the validity of a statement or contract.

Digital Signatures

- MAC: One party generates MAC, one party verifies integrity.
- Digital signatures: One party generates signature, many parties can verify.
- Digital Signature: a data string which associates a message with some originating entity.
- Digital Signature Scheme:
  - a signing algorithm: takes a message and a (private) signing key, outputs a signature
  - a verification algorithm: takes a (public) key verification key, a message, and a signature
- Provides:
  - Authentication, Data integrity, Non-Repudiation

Digital Signatures and Hash

- Very often digital signatures are used with hash functions, hash of a message is signed, instead of the message.
- Hash function must be:
  - Pre-image resistant
  - Weak collision resistant
  - Strong collision resistant



RSA Signatures

Key generation (as in RSA encryption):

- Select 2 large prime numbers of about the same size, p and q
- Compute n = pq, and $\Phi = (q - 1)(p - 1)$
- Select a random integer e, $1 < e < \Phi$, s.t. $\gcd(e, \Phi) = 1$
- Compute d, $1 < d < \Phi$ s.t. $ed \equiv 1 \bmod \Phi$

Public key: (e, n) **used for verification**

Secret key: d, **used for generation**

RSA Signatures (cont.)

Signing message M

- Verify $0 < M < n$
- Compute $S = M^d \bmod n$

Verifying signature S

- Use public key (e, n)
- Compute $S^e \bmod n = (M^d \bmod n)^e \bmod n = M$

Note: in practice, a hash of the message is signed and not the message itself.