IoTSSC – IoT Security Part II

Image: xkcd

Symmetric key encryption

Symmetric-key algorithms use the same key for encryption and decryption

Key typically provisioned at device manufacturing

Block ciphers are most common

(take n-bit blocks of plain text and convert to n-bit blocks of ciphertext)

Multiple stages break the input into groups and perform a set of operations

Substitutions and permutations;

Split, passed through round functions, XORed

Feistel cipher

- Encryption and decryption are very similar/identical
- Message split into blocks and iterated with a ‘round’ function *F* that operates on half of the block with round sub-keys
- Round function does not have to be invertible
- Several schemes employ this, including
  - DES (Data Encryption Standard) and
  - XTEA (eXtended Tiny Encryption Algorithm)

Feistel cipher

1) Divide block into two parts of equal size (L_0, R_0)

2) At each round *i*

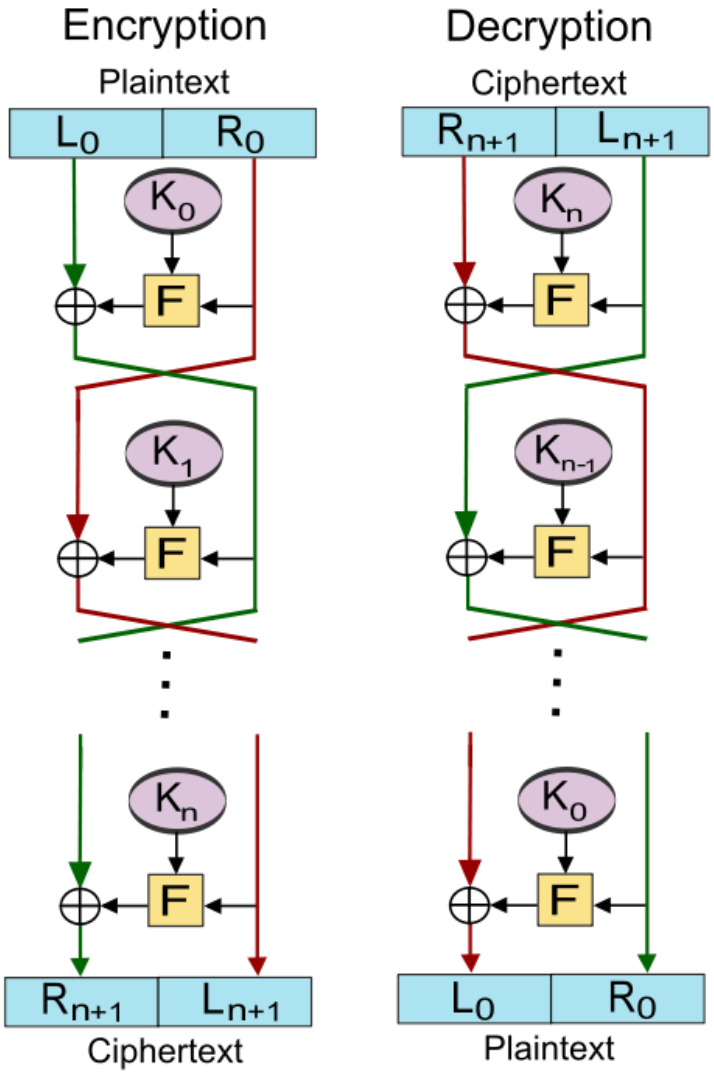
$L_{i+1} = R_i$

$R_{i+1} = L_i \oplus F(R_i, K_i)$

3) Ciphertext is

(R_{n+1}, L_{n+1})

Decryption uses the sub-keys in reverse order



Example

Plaintext: 0xAABBCCDD

Round function: $F(R_i,K_i) = K_i \oplus (R_i >> 2)$

($\oplus$ is the exclusive OR operator and >> is circular shift right)

64-bit key, sub-keys K_0 = 0x1A2B, K_1 = 0x3F07

L_0 = 0xAABB, R_0=0xCCDD

L_1 = , R_1=

Example

Plaintext: 0xAABBCCDD

Round function: $F(R_i,K_i) = K_i \oplus (R_i >> 2)$

($\oplus$ is the exclusive OR operator and >> is circular shift right)

64-bit key, sub-keys K_0 = 0x1A2B, K_1 = 0x3F07

L_0 = 0xAABB, R_0=0xCCDD

$L_1 = R_0$=0xCCDD, $R_1 = L_0 \oplus (K_0 \oplus (R_0 >> 2))$

Example

Plaintext: 0xAABBCCDD

Round function: $F(R_i,K_i) = K_i \oplus (R_i >> 2)$

($\oplus$ is the exclusive OR operator and >> is circular shift right)

64-bit key, sub-keys K_0 = 0x1A2B, K_1 = 0x3F07

L_0 = 0xAABB, R_0=0xCCDD

$L_1 = R_0$=0xCCDD, $R_1 = L_0 \oplus (K_0 \oplus (R_0 >> 2))$

$= 0xAABB \oplus (0x1A2B \oplus 0x7337)$

$= 0xAABB \oplus 0x691C = 0xC3A7$

Example

Plaintext: 0xAABBCCDD

Round function: $F(R_i,K_i) = K_i \oplus (R_i >> 2)$

($\oplus$ is the exclusive OR operator and >> is circular shift right)

64-bit key, sub-keys K_0 = 0x1A2B, K_1 = 0x3F07

L_0 = 0xAABB, R_0=0xCCDD

$L_1 = R_0$=0xCCDD, $R_1 = L_0 \oplus (K_0 \oplus (R_0 >> 2))$

$= 0xAABB \oplus (0x1A2B \oplus 0x7337)$

$= 0xAABB \oplus 0x691C = 0xC3A7$

$L_2 = R_1$= 0xC3A7, $R_2 = L_1 \oplus (K_1 \oplus (R_1 >> 2))$

…

Advanced Encryption Standard (AES)

- Employs a Rijndael cypher (substitution-permutation network)
- Fixed block size (128 bits) arranged in a 4x4 column-major order matrix of bytes
- Key size determines the number of rounds
- Key expanded into round keys through a Rijndael key schedule (byte rotation, exponentiation of 2 to user specified value, S-box)

AES operation

At each round (except last):

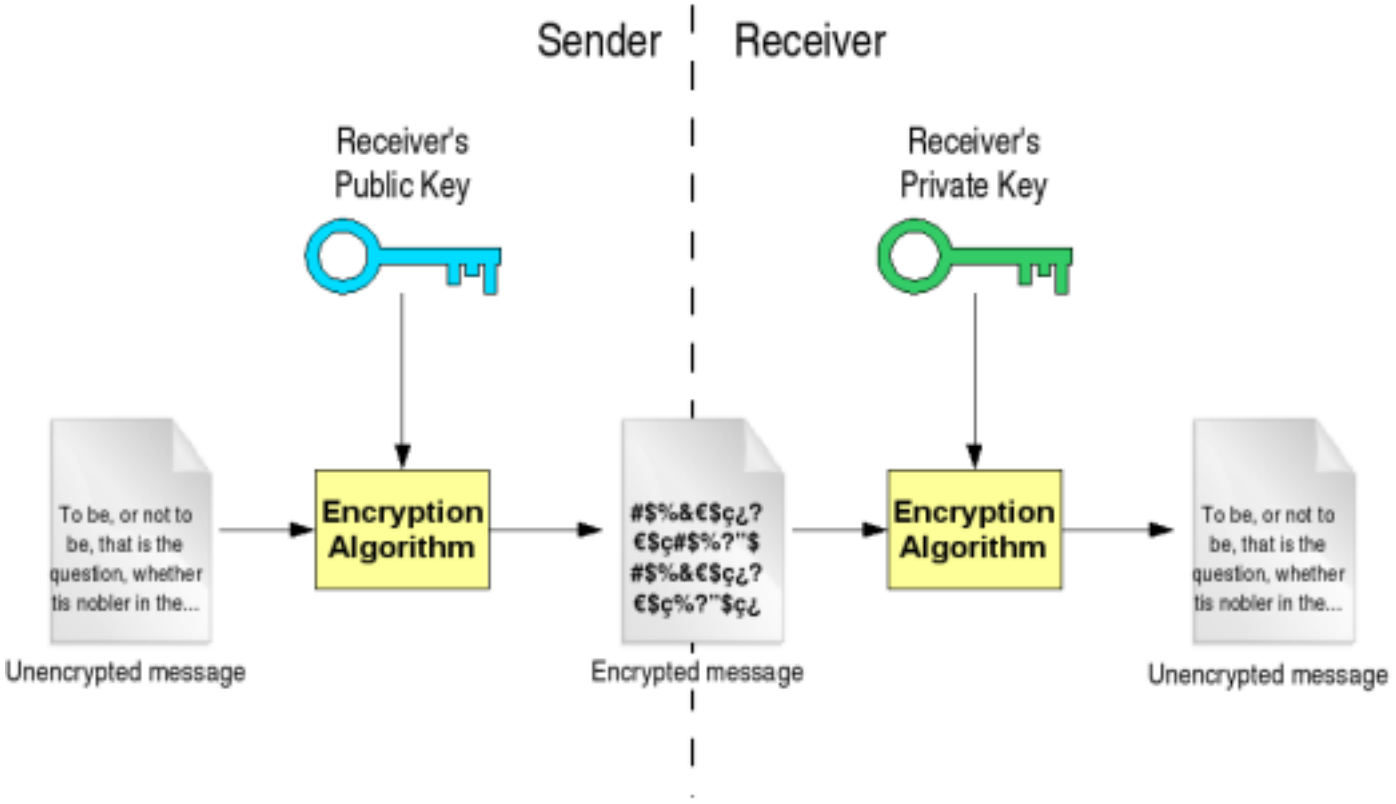
1. SubBytes non-linear byte substitution using a lookup table (S-box) – first 4 bytes indicate row, last 4-bytes indicate column – Galois field GF(2^8)
2. ShiftRows – rotate left bytes on each row *i* with *i-1* positions
3. MixColumns – multiply in GF(2^8) a matrix with each column (diffusion – obscure connection between key and ciphertext)
4. AddRoundKey – XOR between source columns and columns of round key matrix

Columns not mixed at last round.

Asymmetric encryption

- Also known as public-key encryption
- Circumvents problematic key distribution
- Use a public key to encrypt messages (known to anyone) and decrypt with a private key
- Easy to generate public key from private key
- Extremely hard to guess private key

Asymmetric encryption



*The Globus Toolkit 4 Programmer's Tutorial

The RSA cryptosystem (Rivest–Shamir–Adleman)

- Core idea based on prime number factorisation
- Find three very large numbers *e, d, n* such that

$$(m^e)^d = m\ (mod\ n)$$

where it is difficult to find *d* if *e, n,* and *m* are known.

e and *n* will together make the encryption key,

d will be the decryption key, and

m is the message.

RSA operation

1. Randomly choose two large prime numbers p and q
2. Compute $n = p \times q$
3. Compute totient function $\phi(n) = (p - 1) \times (q - 1)$
4. Choose a number e, such that $1<e<\phi(n)$ and e is co-prime with $\phi(n)$ and n

   (only positive factor that divides them is 1)

1. Find d such that $d \times e = 1 \pmod{\phi(n)}$, i.e.

Public key consist of (e,n), private key (d,n)

RSA operation

Encryption: $c = m^e \pmod{n}$

Decryption: $c^d = (m^e)^d = m \pmod{n}$

Example

1. Pick two prime numbers, say p=2 and q=7 (remember: in practice these have to be large!)
2. n = 14
3. $\phi(n) = 1 \times 6 = 6$
4. Choose $1<e<6$ and co-prime with 6 and 14; only e = 5 remains as an option -> public key is (5,14)
5. $5 \times d \pmod 6 = 1$ - > d = 11

$m = 2$ - > $c = 2^5 \pmod{14} = 4$; m' = $4^{11} \pmod{14} = 2$

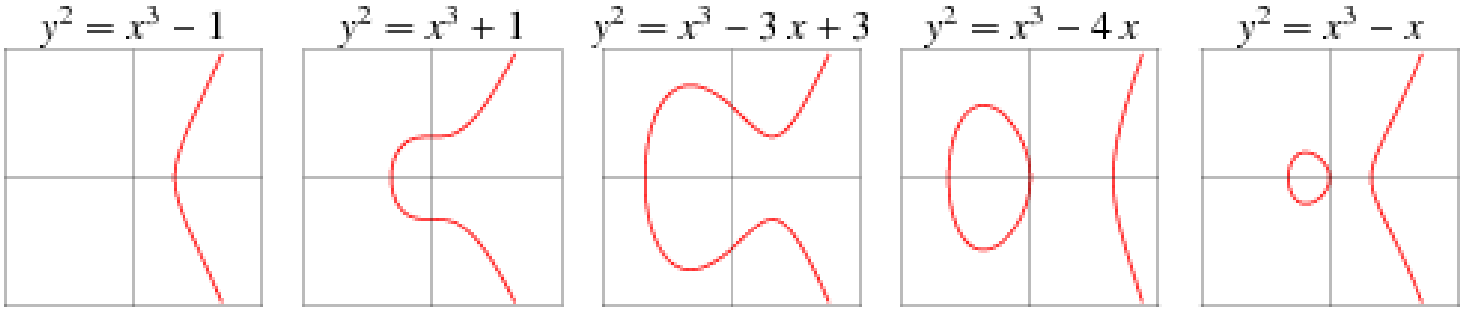
Diffie-Hellman key exchange

Mechanism for exchanging cryptographic keys over a public channel in a secure way

1. The two parties agree publicly on two prime numbers (a generator *g* and a modulus *p*)
2. One party selects a secret prime number *a*, computes g^a mod *p,* and send the result *A* to the other party.
3. The other party does the same with a secret number *b*, computes $B = g^b$ mod *p,* and sends this back.
4. First party computes A^b mod p, second party computes B^a mod p.
5. The result will be the same, i.e. g^{ab} mod p (shared secret)

Elliptic curve encryption

- Again public key cryptography
- Based on the structure of elliptic curves ($y^2=x^3+ax+b$) over finite fields, on which group operations are performed
- Key advantage: smaller keys than with RSA required to provide equivalent security



Elliptic curve encryption

- Key idea: scalar multiplication is a one way function (easy to compute Q = k x P), but very hard to find k when P and Q known.
- Generator – point on the curve that generates a cycling group of n points by repeated additions (order)
- k – smaller number such that k x G = ∞
- Co-factor h = (total number of points on curve)/n should be as close to 1 as possible.
- Field parameter p (modulo)
- Domain parameters {p, a, b, G, n, h}

Key exchange

- Alice sends A = αG, Bob sends B=βG
- Alice computes αB
- Bob computes βB
- Result is the same for both: $\alpha\beta$G
- Shared key established

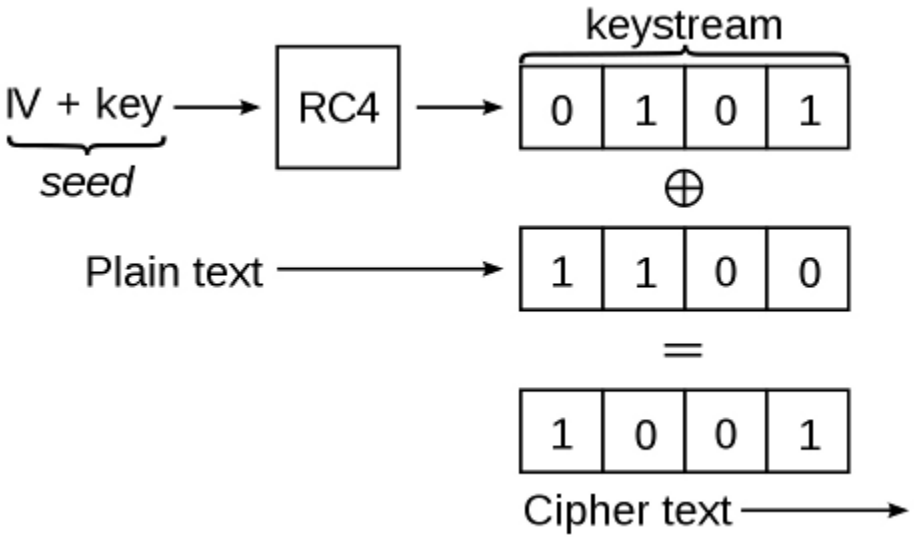
Securing wireless communications

- We talked about the principles of symmetric and asymmetric encryption
- Data confidentiality is particularly important in wireless communications, since the transmission medium is inherently broadcast (anyone can listen)
- Wi-Fi is today the most popular wireless technology and employed in a range of IoT scenarios.
- How is Wi-Fi communication secured?

Wired Equivalent Privacy (WEP)

- Introduced in 1997 with the goal of providing confidentiality comparable to that in wired networks
- In essence WEP employs a stream cipher to hide plaintext information. Key idea:
  - Each bit of the message is XOR-ed with a bit of a pseudorandom stream, called keystream
  - Keystream generated using shift registers initialised with a random seed
  - Vulnerable to attacks if the same seed used twice
  - Keystream obtained by concatenating 40/104-bit key with 24-bit initialisation vector (IV)
- CRC-32 used for integrity check

WEP's weakness



- IVs transmitted in plain text (to avoid repetitions)
- With 24-bit IVs, repetitions do not occur that often (attacks by passively sniffing packets take long)
- ARP request attack – artificially generate responses that help gathering IVs fast – can crack the key in minutes.

Solutions?

- Implement an additional layer of security on top of the link layer (IPsec, HTTPS, etc.) where possible
- In 2013, IEEE proposed an amendment to the standard (802.11i) that defined Wi-Fi Protected Access (WPA) to replace WEP
- Key idea:
  - Continue using the RC4 cipher, but
  - Employ a per-packet key using the Temporal Key Integrity Protocol (TKIP)
  - Combine the IV with a secret root key and add a rekeying mechanism (change key periodically)

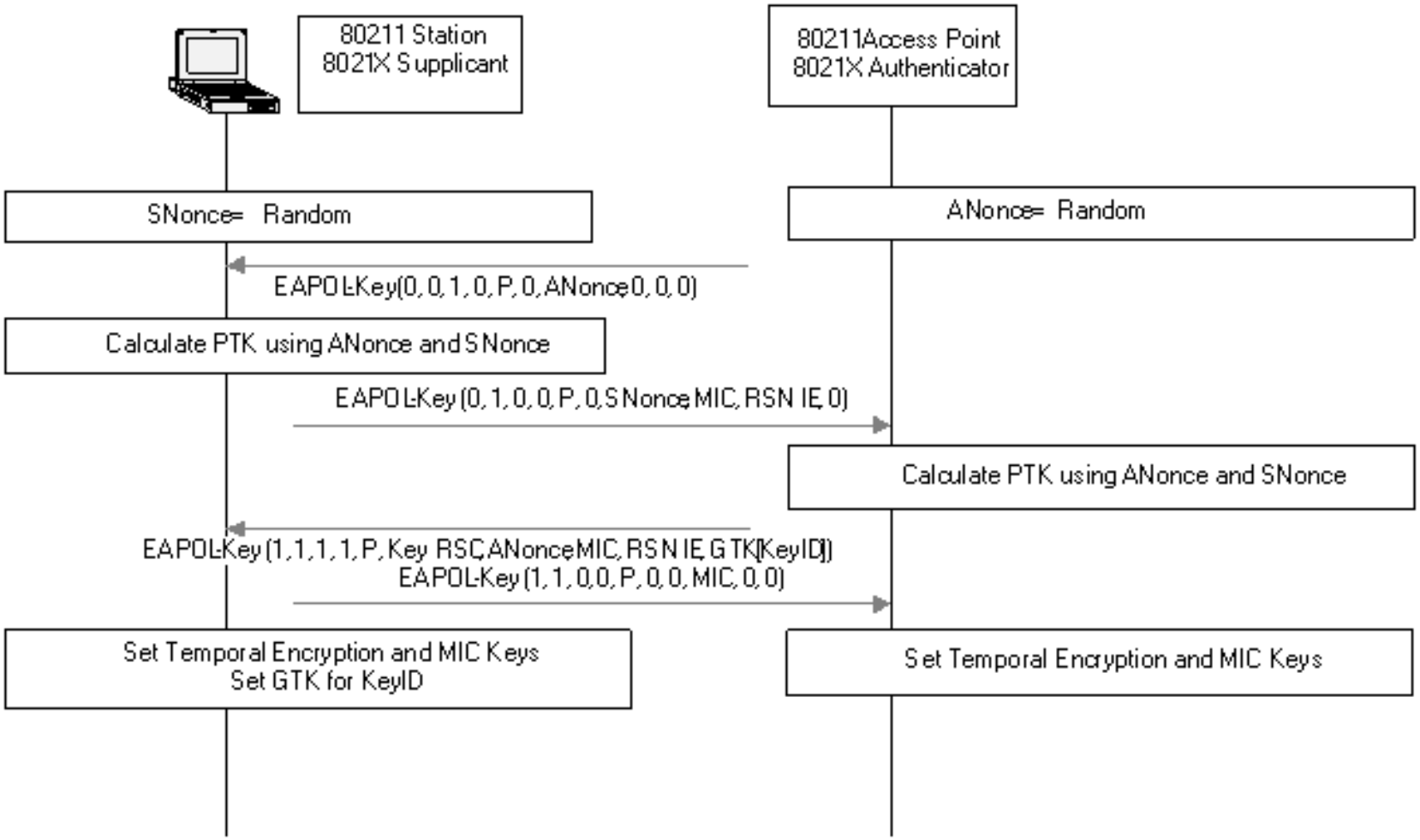
Problems with TKIP

- TKIP implements a 64-bit Message Integrity Check (MIC) – this is exploited by guessing ~14 unknown bytes of ARP packets
- TKIP performs rekeying if two incorrect MIC codes received within 60 seconds.
- Attack still feasible within less than 20 minutes.

WPA2

- RC4 replaced with AES block ciphering
- Initial authentication performed via
  - Pre-shared key (PSK) or
  - Using the Extensible Authentication Protocol (EAP) – Enterprise security, requires an authentication server
- After authentication, a Pairwise Master Key (PMK) is generated, which is derived from a password that is cryptographically hashed.

WPA2 – Four-way handshake



The WPA2 KRACK exploit

- Client and AP prove to each other that they know the PMK, without disclosing it.
- WPA2 mathematically proven safe.
- Used without issues for almost 15years.
- Until the end of 2017 when the KRACK attack was published - Linux and Android susceptible.
- Flaw: Client reinstalls an all-zero encryption key when the third message is received a second time.
- A man-in-the-middle needs to be set-up for this

WPA3

- KRACK fixed in WPA3 (June 2018)
- Additional features
  - Opportunistic Wireless Encryption – Replaces PSK with Diffie-Hellman key exchange, specific for each device
  - Simultaneous Authentication of Equals (SAE) – more secure pre-shared keys, yet easy to use (weak passwords)
  - Secure connection for devices with limited UI (e.g. without displays) - IoT
  - 192-bit encryption in enterprise settings
  - Forward secrecy


Questions?