Crypto Basics

Recent block cipher: AES
Public Key Cryptography
Public key exchange: Diffie-Hellmann
Homework suggestion
What is a cryptosystem?

- $K = \{0,1\}^l$
- $P = \{0,1\}^m$
- $C' = \{0,1\}^n, C \subseteq C'$

- $E: P \times K \rightarrow C$
- $D: C \times K \rightarrow P$

- $\forall p \in P, k \in K: D(E(p,k),k) = p$
  - It is *infeasible* to find inversion $F: P \times C \rightarrow K$

Lets start again!
This time in English ... .
What is a cryptosystem?

- A pair of algorithms that take a key and convert plaintexts to ciphertexts and backwards later
  - **Plaintext**: text to be protected
  - **Ciphertext**: should appear like random
- Requires sophisticated math!
  - **Do not try to design your own algorithms!**
The language of cryptography

- Symmetric or secret key crypto:
  sender and receiver keys are identical and secret

- Asymmetric or Public-key crypto:
  encrypt key public, decrypt key secret
Strength of DES???

- 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- Brute force search looked hard in the seventies/eighties
- Recent advances have shown it is possible
  - in 1997 on Internet in a few months
  - in 1998 on dedicated h/w (EFF) in a few days
  - in 1999 above combined in 22hrs!
- Now have several analytic attacks on DES
  - these utilise some deep structure of the cipher
    - by gathering information about encryptions
    - can eventually recover some/all of the sub-key bits
    - if necessary then exhaustively search for the rest
- Generally these are statistical attacks
- Include
  - differential cryptanalysis
  - linear cryptanalysis
  - related key attacks
- Thus, consider alternatives to DES – AES
AES – Advanced Encryption Standard
Origins

● A replacement for DES was needed
  ○ have theoretical attacks that can break it
  ○ have demonstrated exhaustive key search attacks

● Triple-DES possible – but slow, has small blocks

● US NIST issued a call for ciphers in 1997

● 15 candidates accepted in Jun 98

● 5 were shortlisted in Aug-99

● Rijndael was selected as the AES in Oct-2000

● Accepted as FIPS PUB 197 standard in Nov-2001
AES Requirements

- Private key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger & faster than Triple-DES
- Active life of 20-30 years (+ archival use)
- Full open specification & design details
- Good C, Java, and hw implementations
- NIST releases all submissions & unclassified analyses
AES Evaluation Criteria

- Initial criteria:
  - security – effort for practical cryptanalysis
  - cost – in terms of computational efficiency
  - algorithm & implementation characteristics

- Final criteria
  - general security
  - ease of software & hardware implementation
  - implementation attacks
  - flexibility (in en/decrypt, keying, other factors)
AES Shortlist

- After testing and evaluation, shortlist in Aug-99:
  - MARS (IBM) - complex, fast, high security margin
  - RC6 (USA) - very simple, very fast, low security margin
  - Rijndael (Belgium) - clean, fast, good security margin
  - Serpent (Euro) - slow, clean, very high security margin
  - Twofish (USA) - complex, very fast, high security margin

- Then subject to further analysis & comment

- Saw contrast between algorithms with
  - few complex rounds versus many simple rounds
  - which refined existing ciphers vs. new proposals
The AES Cipher - Rijndael

- Designed by Rijmen-Daemen in Belgium

- Has 128/192/256 bit keys, 128 bit data or block size

- An iterative rather than feistel cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round

- Designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity
Rijndael

Data block of 4 columns of 4 bytes is state

Key is expanded to array of words

Has 9/11/13 rounds in which state undergoes:
  - byte substitution (1 S-box used on every byte)
  - shift rows (permute bytes between groups/columns)
  - mix columns (subs using matrix multiply of groups)
  - add round key (XOR state with key material)
  - view as alternating XOR key & scramble data bytes

Initial XOR key material & incomplete last round

With fast XOR & table lookup implementation
Rijndael
Rijndael
**Byte Substitution**

- a simple substitution of each byte

- uses one table of 16x16 bytes containing a permutation of all 256 8-bit values

- each byte of state is replaced by byte indexed by row (left 4-bits) & column (right 4-bits)
  - eg. byte \{95\} is replaced by byte in row 9 column 5
  - which has value \{2A\}

- S-box constructed using defined transformation of values in GF(2^8)

- designed to be resistant to all known attacks
Byte Substitution

\[
\begin{array}{ccccccccccccccccc}
16 & 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline
0 & 63 & 7c & 77 & 7b & f2 & f8 & 6f & c5 & 80 & 89 & 98 & d7 & ab & a0 & 8b &
1 & c0 & ca & af & 82 & f3 & 00 & 09 & af & 5c & 9e & 73 & 67 & a8 & fe & 77
2 & cb & d1 & 62 & 87 & 7e & c3 & e8 & ea & 07 & 6f & 7c & 58 & 91 & 7a & e0
3 & 73 & b0 & d0 & c7 & #1 & 40 & 05 & 75 & 9f & af & b6 & d7 & 6b & 8e & d9
4 & 1a & 92 & f0 & b8 & c9 & 0f & 4d & ec & e2 & 46 & be & 17 & 4d & e9 &
5 & d9 & e9 & f2 & da & f4 & e8 & c5 & 8c & c7 & 44 & 1d & 76 & 2b & 9c & b6
6 & 4f & ed & 4b & 68 & 5e & 09 & 65 & 33 & 17 & f0 & 95 & 26 & 64 & 44 & 96
7 & d2 & f1 & 39 & b3 & d3 & 92 & 1a & 4b & fc & 5c & 3e & 04 & 8c & a6 & 22
8 & 5a & 6b & c3 & d0 & fc & f8 & 8c & 77 & c6 & 04 & 38 & 1b & 37 & 09 & 97
9 & 5c & 93 & 6a & f9 & 1a & 85 & a5 & 0c & e3 & 1b & f5 & c0 & 27 & 38 & 02
10 & 75 & 01 & b7 & 9b & 47 & 2b & 0a & 17 & 74 & 08 & 9c & 26 & 3d & 32 & 05
11 & 6e & 3b & f7 & 95 & 62 & 87 & 9a & 7b & 0c & 97 & 13 & 5a & 62 & 00 & 82
12 & af & 09 & 98 & f6 & 0a & 20 & 05 & 23 & 10 & 36 & 98 & 5c & 6d & 8e & 69
13 & df & 08 & f3 & 00 & 3e & 73 & 7a & f7 & 54 & 50 & 39 & 48 & 9b & 13 & 3c
15 & 60 & 16 & 32 & 7f & 01 & 6e & 62 & 5c & 25 & 65 & 49 & 1d & 23 & 0a & 61
\end{array}
\]

\[S_0 \ 0 \hline \ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15
\]

S-box

\[
\begin{array}{cccc}
S_0 & S_0 & S_0 & S_0 \\
S_1 & S_1 & S_1 & S_1 \\
S_2 & S_2 & S_2 & S_2 \\
S_3 & S_3 & S_3 & S_3
\end{array}
\]
Shift Rows

- a circular byte shift in each row:
  - 1\textsuperscript{st} row is unchanged
  - 2\textsuperscript{nd} row does 1 byte circular shift to left
  - 3\textsuperscript{rd} row does 2 byte circular shift to left
  - 4\textsuperscript{th} row does 3 byte circular shift to left

- decrypt inverts using shifts to right

- since state is processed by columns, this step permutes bytes between the columns
Shift Rows
Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly m(x) = x^8 + x^4 + x^3 + x + 1

\[
\begin{bmatrix}
02 & 03 & 01 & 01 \\
01 & 02 & 03 & 01 \\
01 & 01 & 02 & 03 \\
03 & 01 & 01 & 02 \\
\end{bmatrix}
\begin{bmatrix}
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\
\end{bmatrix}
= 
\begin{bmatrix}
s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\
s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\
s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\
s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\
\end{bmatrix}
\]
Mix Columns
Mix Columns

- can express each col as 4 equations
  - to derive each new byte in col

- decryption requires use of inverse matrix
  - with larger coefficients, hence a little harder

- have an alternate characterisation
  - each column a 4-term polynomial
  - with coefficients in GF(2^8)
  - and polynomials multiplied modulo (x^4+1)
Add Round Key

- XOR state with 128-bits of the round key
- again processed by column (though effectively a series of byte operations)
- inverse for decryption identical
  - since XOR own inverse, with reversed keys
- designed to be as simple as possible
  - a form of Vernam cipher on expanded key
  - requires other stages for complexity / security
Add Round Key

\[ S_{0,0} \quad S_{0,1} \quad S_{0,2} \quad S_{0,3} \]
\[ S_{1,0} \quad S_{1,1} \quad S_{1,2} \quad S_{1,3} \]
\[ S_{2,0} \quad S_{2,1} \quad S_{2,2} \quad S_{2,3} \]
\[ S_{3,0} \quad S_{3,1} \quad S_{3,2} \quad S_{3,3} \]

\[ \oplus \]

\[ W_i \quad W_{i+1} \quad W_{i+2} \quad W_{i+3} \]

= 

\[ s'_{0,0} \quad s'_{0,1} \quad s'_{0,2} \quad s'_{0,3} \]
\[ s'_{1,0} \quad s'_{1,1} \quad s'_{1,2} \quad s'_{1,3} \]
\[ s'_{2,0} \quad s'_{2,1} \quad s'_{2,2} \quad s'_{2,3} \]
\[ s'_{3,0} \quad s'_{3,1} \quad s'_{3,2} \quad s'_{3,3} \]
AES Round
AES Key Expansion

- takes 128-bit (16-byte) key and expands into array of 44/52/60 32-bit words

- start by copying key into first 4 words

- then loop creating words that depend on values in previous & 4 places back
  - in 3 of 4 cases just XOR these together
  - 1st word in 4 has rotate + S-box + XOR round constant on previous, before XOR 4th back
AES Key Expansion

\[
\begin{array}{cccc}
 k_0 & k_4 & k_8 & k_{12} \\
 k_1 & k_5 & k_9 & k_{13} \\
 k_2 & k_6 & k_{10} & k_{14} \\
 k_3 & k_7 & k_{11} & k_{15} \\
\end{array}
\]

\[
\begin{array}{cccc}
 w_0 & w_1 & w_2 & w_3 \\
\end{array}
\]

\[
\begin{array}{cccc}
 w_4 & w_5 & w_6 & w_7 \\
\end{array}
\]

\[g\]
Key Expansion Rationale

- designed to resist known attacks

- design criteria included
  - knowing part key insufficient to find many more
  - invertible transformation
  - fast on wide range of CPU’s
  - use round constants to break symmetry
  - diffuse key bits into round keys
  - enough non-linearity to hinder analysis
  - simplicity of description
AES Decryption

- AES decryption is not identical to encryption since steps done in reverse

- but can define an equivalent inverse cipher with steps as for encryption
  - but using inverses of each step
  - with a different key schedule

- works since result is unchanged when
  - swap byte substitution & shift rows
  - swap mix columns & add (tweaked) round key
AES Decryption

![Diagram of AES Decryption process]

Key

Expand key

w[0, 3]

Inverse mix cols

w[4, 7]

Round 10

Inverse sub bytes

Inverse shift rows

Inverse mix cols

Add round key

Round 9

w[36, 39]

Inverse mix cols

Round 1

Inverse sub bytes

Inverse shift rows

Inverse mix cols

Add round key

Ciphertext

Add round key

Inverse sub bytes

Inverse shift rows

Inverse mix cols

 Plaintext
Implementation Aspects

- can efficiently implement on 8-bit CPU
  - byte substitution works on bytes using a table of 256 entries
  - shift rows is simple byte shift
  - add round key works on byte XOR’s
  - mix columns requires matrix multiply in GF(2^8) which works on byte values, can be simplified to use table lookups & byte XOR’s
Implementation Aspects

- can efficiently implement on 32-bit CPU
  - redefine steps to use 32-bit words
  - can precompute 4 tables of 256-words
  - then each column in each round can be computed using 4 table lookups + 4 XORs
  - at a cost of 4Kb to store tables

- designers believe this very efficient implementation was a key factor in its selection as the AES cipher
Security of AES

- No crypt-analytical weaknesses for 10 round version
  - Shorter round versions of AES are provably less secure.
- Optimized implementation easiness produced several
  - Fundamental implementation security issues.
  - Difficulties for fast and secure implementations in hw and sw.
Homework suggestion (for next lecture)

- Please read the following papers:

---

**Figure 4:** Evolution of the cache versus time. Each horizontal line represents the state of the cache lines (represented by a point) at a given time. Different AES encryption are clearly visible in the center. The brighter a point, the longer the time to access its corresponding cache line.
Private-Key Cryptography

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- hence does not protect sender from receiver forging a message & claiming is sent by sender
Public-Key Cryptography

❖ probably most significant advance in the 3000 year history of cryptography

❖ uses two keys –
  ❖ a public & a private key

❖ asymmetric since parties are not equal

❖ uses clever application of number theoretic concepts to function

❖ complements rather than replaces private key crypto
Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender

- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community
Public-Key Cryptography

- public-key/two-key/asymmetric cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**

- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures
Public key cryptography
Public-Key Characteristics

- Public-Key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm & encryption key
  - it is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
  - either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
Public-Key Applications

- can classify uses into 3 categories:
  - encryption/decryption (provide secrecy)
  - digital signatures (provide authentication)
  - key exchange (of session keys)

- some algorithms are suitable for all uses, others are specific to one
Security of Public Key Schemes

- like private key schemes brute force **exhaustive search** attack is always theoretically possible
- but keys used are too large (>512bits)
- security relies on a **large enough** difference in difficulty between easy (en/decrypt) and hard (crypt-analyse) problems
- more generally the **hard** problem is known, but is made hard enough to be impractical to break
- requires the use of **very large numbers**
- hence is **slow** compared to private key schemes
Public Key Cryptography

**Symmetric key crypto**
- Requires sender, receiver to know shared secret key
- Q: how to agree on key in first place (particularly if never “met”)?
- Q: what if key is stolen?
- Q: what if you run out of keys?
- Q: what if A doesn’t know she wants to talk to B?

**Public key cryptography**
- Radically different approach [Diffie-Hellman76, RSA78]
- Sender, receiver do *not* share secret key
- Encryption key *public* (known to *all*)
- Decryption key private (known only to receiver)
- Allows parties to communicate without prearrangement
Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
- eg. 2, 3, 5, 7 are prime, 4, 6, 8, 9, 10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

  2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61
  67  71  73  79  83  89  97 101 103 107 109 113 127 131
  137 139 149 151 157 163 167 173 179 181 191 193
  197 199
Relatively Prime Numbers & GCD

- Two numbers $a, b$ are **relatively prime** if they have **no common divisors** apart from 1.
  - Eg. 8 & 15 are relatively prime since factors of 8 are 1, 2, 4, 8 and of 15 are 1, 3, 5, 15 and 1 is the only common factor.

- Conversely, can determine the greatest common divisor by comparing their prime factorizations and using least powers.
  - Eg. $300 = 2^1 \times 3^1 \times 5^2$  $18 = 2^1 \times 3^2$ hence $\text{GCD}(18, 300) = 2^1 \times 3^1 \times 5^0 = 6$
Fermat's Theorem

\[ a^{p-1} = 1 \pmod{p} \]

- where \( p \) is prime and \( \gcd(a, p) = 1 \)

- also known as Fermat’s Little Theorem

- also \( a^p = p \pmod{p} \)

- useful in public key and primality testing
Euler Totient Function $\phi(n)$

- when doing arithmetic modulo $n$
- **complete set of residues** is: $0..n - 1$
- **reduced set of residues** is those numbers (residues) which are relatively prime to $n$
  - eg for $n=10$,
    - complete set of residues is $\{0,1,2,3,4,5,6,7,8,9\}$
    - reduced set of residues is $\{1,3,7,9\}$
- number of elements in reduced set of residues is called the **Euler Totient Function** $\phi(n)$
Euler Totient Function $\varphi(n)$

- to compute $\varphi(n)$ need to count number of residues to be excluded

- in general need prime factorization, but
  - for $p$ ($p$ prime) $\varphi(p) = p-1$
  - for $p.q$ ($p, q$ prime) $\varphi(pq) = (p-1)(q-1)$

- eg.
  $\varphi(37) = 36$
  $\varphi(21) = (3-1)(7-1) = 2 \times 6 = 12$
Euler's Theorem

- A generalisation of Fermat's Theorem
- $a^{\phi(n)} = 1 \pmod{n}$
  - For any $a, n$ where $\gcd(a, n) = 1$

**Example:**

- $a = 3; n = 10; \quad \phi(10) = 4;$
  - Hence $3^4 = 81 = 1 \pmod{10}$
- $a = 2; n = 11; \quad \phi(11) = 10;$
  - Hence $2^{10} = 1024 = 1 \pmod{11}$
Primitive Roots

- from Euler’s theorem have $a^{\phi(n)} \mod n = 1$
- consider $a^m = 1 \mod n$, $\gcd(a, n) = 1$
  - must exist for $m = \phi(n)$ but may be smaller
  - once powers reach $m$, cycle will repeat
- if smallest is $m = \phi(n)$ then $a$ is called a primitive root or generating element
- if $p$ is prime, then successive powers of $a$ "generate" the group $\mod p$
- these are useful but relatively hard to find
Discrete Logarithms

- the inverse problem to exponentiation is to find the discrete logarithm of a number modulo p
- that is to find \( x \) such that \( y = g^x \mod p \)
- this is written as \( x = \log_g y \mod p \)
- if \( g \) is a primitive root then it always exists, otherwise it may not, eg.
  - \( x = \log_3 4 \mod 13 \) has no answer
  - \( x = \log_2 3 \mod 13 = 4 \) by trying successive powers
- whilst exponentiation is relatively easy, finding discrete logarithms is generally a hard problem
Public-Key distribution of Secret Keys

- use previous methods to obtain public-key
- can use for secrecy or authentication
- but public-key algorithms are slow
- so usually want to use private-key encryption to protect message contents
- hence need a session key
- have several alternatives for negotiating a suitable session
Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now known that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products
Diffie-Hellman Key Exchange

- a public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) – seems easy at first sight
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard
Diffie-Hellman Setup

- all users agree on global parameters:
  - large prime integer or polynomial \( q \)
  - \( a \) being a primitive root mod \( q \)

- each user (eg. A) generates their key
  - chooses a secret key (number): \( x_A < q \)
  - compute their **public key**:
    \[ y_A = a^{x_A} \mod q \]

- each user makes public that key \( y_A \)
Diffie-Hellman Key Exchange

User A

Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \mod q$

Calculate $K = (Y_B)^{X_A} \mod q$

User B

Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \mod q$;
Calculate $K = (Y_A)^{X_B} \mod q$
Diffie-Hellman Key Exchange

- shared session key for users A & B is $K_{AB}$:
  $$K_{AB} = a^{x_A} \cdot x_B \mod q$$
  $$= y_A^{x_B} \mod q \quad \text{(which B can compute)}$$
  $$= y_B^{x_A} \mod q \quad \text{(which A can compute)}$$

- $K_{AB}$ is used as session key in private-key encryption scheme between Alice and Bob

- if Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-keys

- attacker needs an $x$, thus must solve discrete log, logarithm modulo $q$, i.e., compute $x_A$ from $y_A = a^{x_A}$
Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $a=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=3^{97} \mod 353 = 40$ (Alice)
  - $y_B=3^{233} \mod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \mod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \mod 353 = 40^{233} = 160$ (Bob)
Key Exchange Protocols

- users could create random private/public D-H keys each time they communicate

- users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them

- both of these are vulnerable to a meet-in-the-Middle Attack

- authentication of the keys is needed
  - Next lectures more on this!
Summary

- Have considered:
  - Details of Rijndael – the AES cipher
  - Principle of Public Key Cryptography
  - Number Theory basics
  - Diffie-Hellmann Key exchange