Stream ciphers
- Encrypt chars/bits one at a time
- Assume XOR w the key, need long key to be secure

Keystream generators (pseudo-random key)
- Synchronous and self-synchronizing
- Linear congruential generators
- LFSRs

Block ciphers
- Multiple rounds of substitution and transposition
- Modes: ECB, CBC, CFB, OFB

Public Key Cryptography
Everyone has two keys:
- Public key K1 that everyone knows
- Private key K2 that only he knows
- Encryption algorithm and key properties ensure that
  \[ D_{K2}(E_{K1}(M)) = M \]

Galois Field GF(n)
- All operations are on numbers 0,...,n–1
- Observe all operations in Galois Field GF(n)
  - \( a = b \mod n \) if \( 3k, a = k'n + b \)
  - e.g. 26 \( \mod 16 = 10 \) so
  - 26 = 10 \( \mod 16 \)
  - Modulo operation (modular reduction) can be performed at any point, e.g.
    \( (a + b) \mod n = ((a \mod n) + (b \mod n)) \mod n \)

How many operations are needed to calculate \( a^x \)?
- Exponentiation can be performed very efficiently (addition chaining):
  - We want to calculate \( a^x \mod n \)
  - Write \( x \) as a binary number, result = 1
  - Traverse \( x \) from left to right
    - If digit is 1, result = result\(^2\) * a
    - If digit is 0, result = result\(^2\)
    - Perform modular reduction often to keep result small. This is cheap if \( n = 2^m \)

Example
\[
\begin{align*}
4^{19} \mod 23 &= 19 = 10011 \mod 23 = 19 \\
4^{19} \mod 23 &= \text{step1 bit is 1: result = } 1^2 \times 4 = 4 \\
4^{19} \mod 23 &= \text{step2 bit is 0: result = } 4^2 \mod 23 = 16 \\
4^{19} \mod 23 &= \text{step3 bit is 0: result = } 16^2 \mod 23 = 256 \mod 23 = 3 \\
4^{19} \mod 23 &= \text{step4 bit is 1: result = } 3^2 \times 4 \mod 23 = 36 \mod 23 = 13 \\
4^{19} \mod 23 &= \text{step5 bit is 1: result = } 13^2 \times 4 \mod 23 = 676 \mod 23 = 9
\end{align*}
\]
Prime Numbers
- A number \( n \) is prime if it is only divisible by 1 and itself
- Numbers \( x \) and \( y \) are relatively prime if they share no factors greater than 1
  - E.g. 7 and 15 are relatively prime, 9 and 15 are not because they have 3 as common factor

Inverses Modulo a Number
- Multiplicative inverse \( y \) for \( x \) is a number that satisfies:
  \[ x \cdot y \equiv 1 \pmod{n} \]
- In GF(\( n \)) inverse \( y \) for \( x \) modulo \( n \) is a number that satisfies:
  \[ x \cdot y \equiv 1 \pmod{n} \]
- Inverse \( y \) in GF(\( n \)) can be found uniquely if \( x \) and \( n \) are relatively prime, otherwise it cannot be found
- If \( n \) is prime then it is relatively prime to all numbers \( \{0, n-1\} \) and each number has its inverse in GF(\( n \))

Extended Euclidean Algorithm
- How to find an inverse \( y \) for \( x \) mod \( n \)
  \[ x \cdot y \equiv 1 \pmod{n} \]
  \[ x \cdot y - k \cdot n = 1 \]
- Extended Euclidean algorithm will find \( y \) and \( k \) given \( x \) and \( n \)

Extended Euclidean Algorithm
- \( a_i \div \bar{a}_{i-1} = q_{i-1} \), and the remainder \( a_i \)

Extended Euclidean Algorithm
- \( 225/13 = 17 \) remainder 4

Extended Euclidean Algorithm
- \( a_i \div \bar{a}_{i-1} = q_{i-1} \), and the remainder \( a_i \)
Extended Euclidean Algorithm

\[ 13/4 = 3 \text{ remainder } 1 \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & \text{STOP} & \\
1 & 4 & & \\
\end{array}
\]

Extended Euclidean Algorithm

\[ 4/1 = 4 \text{ remainder } 0 \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & 1 & \text{STOP, remainder is 0} \\
1 & 4 & & \\
\end{array}
\]

Extended Euclidean Algorithm

\[ x_{i+1} = x_{i-1} - q_i x_i \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & & \\
1 & 4 & & \\
\end{array}
\]

Extended Euclidean Algorithm

\[ 1-17\times0 = 1 \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & 1 & \\
1 & 4 & & \\
\end{array}
\]

Extended Euclidean Algorithm

\[ 0-3\times1 = -3 = 222 \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & -1 & \\
1 & 4 & 222 & \\
\end{array}
\]

Extended Euclidean Algorithm

\[ y_{i+1} = y_{i-1} - q_i y_i \]

\[
\begin{array}{c|c|c|c}
 a & q & x & y \\
225 & - & 1 & 0 \\
13 & 17 & 0 & 1 \\
4 & 3 & 1 & \\
1 & 4 & 222 & \\
\end{array}
\]
Extended Euclidean Algorithm

\[ 0 \cdot 17 \cdot 1 = 17 + 208 \]

<table>
<thead>
<tr>
<th>a</th>
<th>q</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>225</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>208</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>222</td>
<td></td>
</tr>
</tbody>
</table>

Extended Euclidean Algorithm

\[ 1 - (3 \cdot 208 \bmod 225) = 173 = 52 \]

<table>
<thead>
<tr>
<th>a</th>
<th>q</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>225</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>208</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>222</td>
<td>52</td>
</tr>
</tbody>
</table>

52 is inverse for 13 mod 225
52 \cdot 13 \bmod 225 = 1

Factorization of Large Numbers

- We have seen that exponentiation in \( GF(n) \) can be performed efficiently
- On the other hand, it is hard to factor large numbers
  - Computationally intensive, must use brute-force search
  - Generally factoring time of a large number \( n \) increases exponentially with each binary digit added to \( n \)

Public Key Cryptography (RSA)

- Created by Ron Rivest, Adi Shamir, and Leonard Adleman
- Choose two prime numbers \( p \) and \( q \) of equal length
  - Compute \( n = p \cdot q \), and
  - Euler Totient function: \( \phi(n) = (p-1)(q-1) \)
- Choose public key \( e \) relatively prime to \( \phi(n) \)

Public Key Cryptography (RSA)

- We can easily perform exponentiation in \( GF \) in linear time
- We can calculate \( d \) out of \( e \) and \( n \) in polynomial time using extended Euclidean algorithm (because we know \( p \) and \( q \), we can calculate \( \phi(n) = (p-1)(q-1) \))
- Attacker must factor large number \( n \) to learn \( p \) and \( q \) which is expensive (exponentially with number of bits in \( n \))

Public Key Cryptography (RSA)

- Using extended Euclidean algorithm calculate \( d \) which is inverse of \( e \bmod \phi(n) \)
  - \( d \cdot e = 1 \bmod \phi(n) \)
- Publish \( e \) and \( n \), remember \( d \)
- Encryption:
  - \( E(M) = M^e \bmod n \)
- Decryption:
  - \( D(C) = C^d \bmod n = M^e \bmod n \)
  - \( M^{(e \cdot d - 1) \bmod \phi(n)} \bmod pq = M \)
- There are proofs that the last equation holds
Digital Signatures

- Provide data integrity
  - Can it be done with symmetric systems?
    - Verification requires shared key
    - Doesn’t provide non-repudiation
- Need proof of provenance
  - Hash the data, encrypt with private key
  - Verification uses public key to decrypt hash
  - Provides non-repudiation
    - Why?

One-Way Hash Functions

- Take a variable-length input $M$ and produce fixed-length output (hash value or message digest or message integrity code)
  \[ h = H(M) \]
- The idea is to fingerprint $M$
  - Given $M$ easy to compute $h$
  - Given $h$ very hard to compute $M$
  - One-bit change in $M$ changes on the average half of the bits in $h$
  - Good one-way hash function is collision-free: given $M$ it is very hard to find $M'$ such that $H(M) = H(M')$
  - One-way hash function is public

One-Way Hash Functions

- Divide $M$ into blocks, generate hash value iteratively

\[ \begin{align*}
  h_0 & = M \\
  h_i & = H(h_{i-1}) \\
  \text{Hash value of the whole message is obtained in the last step}
\end{align*} \]

MDS

- Divide $M$ into 512-bit blocks
  - Pad $M$ with string of 1 and many zeros so that it is 64 bit short of multiple of 512
  - Concatenate original length in bits as 64-bit number
- Blocks are processed sequentially
- Last result is hash value for the whole message and is 128-bit long
MD5

- For each message block (512 bits):
  - Variables A, B, C, D are initialized (for first block they are constant values)
    - Each is 32 bits
  - Go through 4 rounds; each round repeats the following operation 16 times:
    - Performs non-linear function on three variables
    - Sums the result, the fourth variable, message subblock (32-bits) and a constant \( K_i \) \( (i \in [1,16]) \)
    - Rotates the result to the left, adds it to one of the variables and replaces this variable
- Output is concatenation of A, B, C, D

MD5 One Block Processing

MD5 One Round = 16 operations

Non-linear functions (in each round one is chosen for the green square):
- \( F(X,Y,Z) = XY \text{ OR} \neg(X) Z \)
- \( G(X,Y,Z) = XZ \text{ OR} Y \neg(Z) \)
- \( H(X,Y,Z) = X \text{ XOR} Y \text{ XOR} Z \)
- \( I(X,Y,Z) = Y \text{ XOR} (X \text{ OR} \neg(Z)) \)

SHA And SHA-1

- Security of MD5 is severely compromised
  - Multiple known attacks
- Secure Hash Algorithm (SHA)
- Similar to MD5 but:
  - Has 20 operations per round
  - Operations are different than in MD5 but along similar lines
  - Each message block (16*32 = 512 bits) is expanded into (80*32 bits)
  - Produces 160-bit hash value
- SHA-1 is an improved version of SHA

DES

- Data Encryption Standard (used to be)
- Block cipher, symmetric
- Works with 64-bit blocks of plaintext and 56-bit key (weak)
  - Superseded by AES

DES

Security of MD5 is severely compromised
- Multiple known attacks
- Secure Hash Algorithm (SHA)
- Similar to MD5 but:
  - Has 20 operations per round
  - Operations are different than in MD5 but along similar lines
  - Each message block (16*32 = 512 bits) is expanded into (80*32 bits)
  - Produces 160-bit hash value
- SHA-1 is an improved version of SHA
**Triple DES**
- Fixes the key length problem by applying DES three times
  - With two or three different keys
  - $E(K_1)D(K_2)E(K_3)$
  - Designed to be backward compatible with DES
- Slow, since encryption/decryption is now performed three times

**DES-X**
- Fixes the key length problem by using a longer key whose portions are XOR-ed with plaintext
  - 64-bits XOR-ed prior to DES
  - 64-bits XOR-ed with DES output
  - Key 56+64+64 bits long
- Same speed as DES but higher security

**AES**
- Advanced Encryption Standard
- Block cipher, symmetric
- Works with 128–bit blocks of plaintext and 128, 192 or 256–bit key
- Operates on 4x4 matrix of bytes called state, all operations over $GF(2^8)$
- Algorithm
  - Key expansion – derive round keys (4x4)
  - AddRoundKey – XOR state with round key
  - Rounds – SubBytes (substitution), ShiftRows (transposition), MixColumns, AddRoundKey
  - Final round – SubBytes, ShiftRows, AddRoundKey

**AES**
- Substitution box
- Multiplication by a known matrix