News/Reminders

- Project 1 due last night, Project 2 due July 31
- Next lecture will be recorded July 31
- Week of July 31 online midterm
  - Should be available by Monday evening, due completed by Wednesday evening
  - 150 min, individual, open note
  - Grades will be available before drop date
Midterm Topics

- Security and risk analysis
- Security current events
- Software vulnerabilities
  - Identify vulnerabilities and how they can be exploited
- Software defenses
  - Isolation/virtualization
  - Stack-based defenses
  - Code-correctness defenses
- Malware and malware defenses

- Symmetric Cryptography
  - Modes of operation
  - DES/AES
  - “Confusion and diffusion”
- Public Key Cryptography
  - Diffie-Hellman/RSA, OAEP
  - assumptions
  - Public-key infrastructure
Cryptography

- Symmetric key cryptography (secret key crypto): sender and receiver keys identical
- Asymmetric key cryptography (public key crypto): encryption key public, decryption key secret (private)
Vernam Ciphers

- **XOR cipher** - encryption and decryption the same, Block of data XOR key
- Vernam’s cipher used a message with a paper tape loop that read off the key
- More modern versions use a pseudorandom number generator (stream cipher)
- **One-time pad** - If key perfectly random AND only used once, then perfect secrecy is assured
  - Drawbacks?
Reusing one-time pads

\[
\begin{array}{ccc}
M_1 & M_2 & E_1 \\
\text{SEND} & \text{CASH} & E_2 \\
\text{E1} & \text{E2} & \text{SEND} \text{ CASH}
\end{array}
\]
Old School Cryptography

- Caesar cipher - shift cipher (each letter replaced by one a fixed length down)
  - “Veni, vidi, vici” -> “Yhql, ylgl, ylel”
- Monoalphabetic substitution : substitute one letter for another
  - S-box - bit level substitution
- Transposition - Permute the order of the message
  - P-box - bit level transposition
Cryptogram exercise

• Shift? Transposition? Substitution?:

  • fqjcb rwjwj bnkhj whxcq
     nawjv nfxdu mbvnu ujbbf nnc
Multiple Round Ciphers

• Multiple rounds of complex ciphers made up of permutations, substitutions, xor, etc

• Examples DES, AES

• DES not so secure because key too short

• Hard to understand, little proof of security (except that if anyone knows how to break they’re not telling)
DES - Data Encryption Standard

- Encrypts by series of substitution and transpositions.
- Based on Feistel Structure
  - Use P-Boxes, S-boxes, and XOR to create “confusion and diffusion”
  - Iterative structure easy to implement in hardware
  - Other Feistel-based algorithms: Blowfish, Camellia, CAST-128, DES, FEAL, ICE, KASUMI, LOKI97, Lucifer, MARS, MAGENTA, MISTY1, RC5, TEA, Triple DES, Twofish, XTEA, GOST_28147-89
- Worldwide standard for more than 20 years.
- Has a history of controversy.
- Designed by IBM (Lucifer) with later help (interference?) from NSA.
- No longer considered secure for highly sensitive applications.
- Replacement standard AES (advanced encryption standard) recently completed.
Data Encryption Standard (DES)
One Round of DES

a. Encryption round

b. Decryption round
DES Function

Every 4 bit chunk is expanded to 6 bits

Image src: DataComm\&nw/4e, Forouzan book
**S-boxes: S1**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>10</th>
<th>11</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>14</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>2</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>15</td>
<td>7</td>
<td>4</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>11</td>
<td>9</td>
<td>5</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1</td>
<td>14</td>
<td>8</td>
<td>13</td>
<td>6</td>
<td>2</td>
<td>11</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
<td>12</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>11</td>
<td>3</td>
<td>14</td>
<td>10</td>
<td>0</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

**Is the table entry from**

\[ S(b_1b_2b_3b_4b_5b_6) \]

**row:** \( b_1b_2 \)

**column:** \( b_3b_4b_5b_6 \)

\[ S(011001) = 6 = 0110 \]
Cryptanalysis

- **Differential cryptanalysis**
  - *chosen plaintext attack* (obtain ciphertexts for sets of plain texts of attacker’s choice) $2^{47}$ plaintext for 16 round DES
  - Use similar sets of plaintexts to trace through the permutations and look for nonrandom patterns in the ciphertexts

- **Linear cryptanalysis**
  - *known plaintext attack* - attacker has samples of plaintext/ciphertext pairs, $2^{43}$ pairs needed for DES
  - Construct linear equations relating keys, plaintext, and ciphertext
DES Security

- DES not too susceptible to differential or linear cryptanalysis
- BUT, 56-bit key is just too short
- EFF's Deep Crack breaks in 56 hours (1998) for $250,000
- distributed.net and Deep Crack 22 hours (1999)
- COPACOBANA FPGA machine $10,000, 6.4 days per key
- 2008 - RIVYERA machine reduced the average time to less than one single day.
Double DES (Multiple Encryption)

- Encrypt twice with two keys

**MEET-IN-THE MIDDLE ATTACK**
- Known plaintext attack (i.e. have crib P1 & C1)

- For all K1 encrypt P1: list all results in Table T

- For each K2 decrypt C1 -> X. If X in T, check K1 & K2 with new crib (P2, C2). If okay then keys found.

- Reduces $2^{112}$ to $2^{56}$ for Double DES, but T is huge!
**Triple DES**

**TRIPLE DES WITH 2 KEYS (EDE2)**

- 3 keys considered unnecessary
- Cost of 2 key attack is thus $2^{112}$
- 2nd Stage is decryption because if $K_2 = K_1$ we gain backward compatibility with Single DES
- Available in PEM (Privacy Enhanced Mail), PGP, and others.

**TRIPLE DES WITH 3 KEYS (EDE3)**

- Preferred by some
Data Lifetime and Security

- List three types of data whose lifetime (amount of time for which confidentiality protection is needed) is approximately one day. List three whose lifetime is closer to one year. List three whose lifetime is closer to one century.
Advanced Encryption Standard (AES)

- 1997 NIST solicited proposals for a new Advanced Encryption Standard (AES) to replace DES.
- 5 algorithms shortlisted: Rijndael won (by Joan Rijmen and Vincent Daemen from Belgium), AES has only minor changes
- NIST estimated that a machine that could break a 56-bit DES key in 1 second would take 149 trillion years to crack a 128-bit AES key
- NSA approved for classified data

<table>
<thead>
<tr>
<th>Size of Data Block</th>
<th>Number of Rounds</th>
<th>Key Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>128 bits</td>
<td>10</td>
<td>128 bits</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>192 bits</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>256 bits</td>
</tr>
</tbody>
</table>
AES overview
Bytes represent finite field elements in GF(2^8), GF means “Galois Field”

Correspond to a 8 term polynomial, with 0 or 1 coefficients.

\[ b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 \]

Example:

\[ x^6 + x^5 + x^3 + x^2 + 1 \] polynomial

\{0110 1101\} binary

6D hex
State

- State is a 4 by 4 array of bytes, initialised (col-by-col) with the 16-byte plaintext block (see below)
- Final value of state is returned as ciphertext

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
</table>

- Bytes of State correspond to finite field elements in GF($2^8$)
- Columns of State correspond to WORDS, i.e. 4-term polynomials with finite field elements in GF($2^8$), as coefficients.
Encrypt Block
(Cipher) // simplified

encrypt (plaintext, roundkey)
    state = plaintext       // note plaintext is 1-dim., state 2-dim.
    state = AddRoundKey (state, roundkey[0])
    for round = 1 to ROUNDS
        state = SubBytes (state)
        state = ShiftRows (state)
        if round < ROUNDS then state = MixColumns (state)
    end
    state = AddRoundKey (state, roundkey[round])
return state       // convert to 1-dim. and return as ciphertext
SubBytes Transformation

Change each byte of state with corresponding byte from SBOX matrix: $\text{SBOX} [X,Y] = \text{AffineTransformation}(\{XY\}^{-1})$

Affine Transformation does a matrix multiplication followed by vector addition
In the ShiftRows step, bytes in each row of the state are shifted cyclically to the left. The number of places each byte is shifted differs for each row.
In the MixColumns step, each column of the state is multiplied with a fixed polynomial $c(x)$. 
Add Round Key

In the AddRoundKey step, each byte of the state is combined with a byte of the round subkey using the XOR operation (⊕).
decrypt (ciphertext, roundkey)
  state = ciphertext       // note cipher is 1-dim., state 2-dim.
  state = AddRoundKey (state, roundkey[ROUNDS])
  for round = ROUNDS-1 to 0
      state = InvShiftRows (state) // ShiftRows inverse mode
      state = InvSubBytes (state)  // SubBytes inverse mode
      state = AddRoundKey (state, roundkey[round])
      if round > 0 then state = InvMixColumns (state)
  end
  return state       // convert to 1D and return as plaintext
Security of AES

• Most successful attacks are side-channel attacks

• Side-channel attacks use weaknesses in the physical implementation of the system, not the algorithm or brute-force keycracking

• D.J. Bernstein showed that delays in encryption time due to cache misses can be used to infer key, demonstrated against a custom remote server using OpenSSL’s AES implementation, Osvik et al showed that local attacks could infer the key in 65 milliseconds

• Theoretical “XSL attack” in 2002 suggests some problems with the mathematics, no practical demonstration and looks thoroughly impractical at this point

• "We have one criticism of AES: we don't quite trust the security...What concerns us the most about AES is its simple algebraic structure...No other block cipher we know of has such a simple algebraic representation. We have no idea whether this leads to an attack or not, but not knowing is reason enough to be skeptical about the use of AES." (Practical Cryptography, 2003, pp56–57) (Schneier, Ferguson)
What if your message is longer than the block?
Electronic CodeBook (ECB)
Problems with ECB
Cipher Block Chain
Other modes of operation

- Propagating cipher-block chaining (PCBC) - propagate small changes in ciphertext
- Cipher feedback (CFB) - self-synchronizing stream cipher
- Output feedback mode (OFB) - synchronous stream cipher, errors don’t propagate, but easier to attack
Problems with Symmetric Key Crypto

• Scalability - separate communication between N people requires $N(N-1)/2$ keys

• Key management
  • Key distribution
  • Key storage and backup
  • Key disposal
  • Key change
Applications of Public Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric key cryptography, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (maybe)
    - Secret is stored only at one site

- Digital signatures for authentication
  - Can “sign” a message with private key

- Session Key establishment
  - Exchange messages to create a special session key
  - Then use symmetric key cryptography
Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: $p$ and $g$
  - $p$ is a large prime number, $g$ is a generator of $\mathbb{Z}_p^*$
  - $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}; \forall a \in \mathbb{Z}_p^* \exists i$ such that $a = g^i \mod p$
  - Modular arithmetic (numbers wrap around after they reach $p$)
    - $0 = p \mod p$

**Diagram:**

- Pick secret, random $X$
- $g^x \mod p$
- $g^y \mod p$
- Pick secret, random $Y$
- Compute $k = (g^y)^x = g^{xy} \mod p$
- Compute $k = (g^x)^y = g^{xy} \mod p$
Modular arithmetic exercise

- Come up with addition and multiplication tables for integers mod 4
Why is Diffie-Hellman Secure?

- **Discrete Log (DL) problem:**
  given $g^x \mod p$, it’s hard to extract $x$
  - There is no known efficient algorithm for doing this
  - This is not enough for Diffie-Hellman to be secure! (Why?)

- **Computational Diffie-Hellman problem:**
  given $g^x$ and $g^y$, it’s hard to compute $g^{xy} \mod p$
  - … unless you know $x$ or $y$, in which case it’s easy

- **Decisional Diffie-Hellman (DDH) problem:**
  given $g^x$ and $g^y$, it’s hard to distinguish between $g^{xy} \mod p$ and $g^r \mod p$ where $r$ is random
Properties of Diffie-Hellman

• Assuming DDH problem is hard, Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  • Eavesdropper can’t distinguish between established key and a random value
  • Can use new key for symmetric cryptography
    • Approx. 1000 times faster than modular exponentiation

• Diffie-Hellman protocol (by itself) does not provide authentication
Lec 1 Redux: Diffie-Hellman Handshake

This depends on the hardness of discrete log
(hard to find \( x \) from \( g^x \))
Now both sides have a symmetric key, \( K = g^{xy} \),
Why do we need to encrypt \( g^x \)?
Why do we need \( H(K) \)?
What’s still broken?
Requirements for Public Key Crypto

• Key generation: computationally easy to generate a pair (public key PK, private key SK)
  • Computationally hard to obtain private key SK given only public key PK

• Encryption: given plaintext M and public key PK easy to compute ciphertext C = E_{PK}(M)

• Decryption: given ciphertext C = E_{PK}(M) and private key SK, easy to compute plaintext M
  • Infeasible to compute M from C without SK
  • Even infeasible to learn partial information about M
  • Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M
RSA: Number Theory

- Euler totient function $\phi(n)$, where $n \geq 1$, is the number of integers in the $[1,n]$ interval that are relatively prime to $n$
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1

- Euler’s theorem: if $a \in \mathbb{Z}_n^*$, then $a^{\phi(n)} \equiv 1 \mod n$

- Special case: Fermat’s Little Theorem: if $p$ is prime and $\gcd(a,p)=1$, then $a^{p-1} \equiv 1 \mod p$
RSA Cryptosystem

Key generation:
- Generate large primes p, q (1024 bits? 2048?) use primality test
- Compute n=pq and \( \phi(n) = (p-1)(q-1) \)
- Choose small e, relatively prime to \( \phi(n) \)
  - Typically, e=3 or e=2^{16}+1=65537
- Compute unique d such that ed = 1 mod \( \phi(n) \)
- Public key = (e,n); private key = d

Encryption of m: \( c = m^e \mod n \)
- Modular exponentiation by repeated squaring

Decryption of c: \( c^d \mod n = (m^e)^d \mod n = m \)
Why Decryption Works

- \( e \cdot d = 1 \mod \phi(n) \)
- Thus \( e \cdot d = 1 + k \cdot \phi(n) = 1 + k(p-1)(q-1) \) for some \( k \)
- Let \( m \) be any integer in \( \mathbb{Z}_n \)
- If \( \gcd(m,p) = 1 \), then \( m^{ed} = m \mod p \)
  - By Fermat’s Little Theorem, \( m^{p-1} = 1 \mod p \)
  - Raise both sides to the power \( k(q-1) \) and multiply by \( m \)
  - \( m^{1+k(p-1)(q-1)} = m \mod p \), thus \( m^{ed} = m \mod p \)
  - By the same argument, \( m^{ed} = m \mod q \)
- Since \( p \) and \( q \) are distinct primes and \( p \cdot q = n \),
  - \( m^{ed} = m \mod n \)
Why is RSA Secure?

- RSA problem: given $n=pq$, $e$ such that $\gcd(e,(p-1)(q-1))=1$ and $c$, find $m$ such that $m^e = c \mod n$
  - i.e., recover $m$ from ciphertext $c$ and public key $(n,e)$ by taking $e$th root of $c$
  - There is no known efficient algorithm for doing this

- Factoring problem: given positive integer $n$, find primes $p_1, \ldots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\ldots p_k^{e_k}$
  - If factoring is easy, then RSA problem is easy, but there is no known reduction from factoring to RSA
  - It may be possible to break RSA without factoring $n$
Caveats

- Don’t use RSA directly
- $e = 3$ is a common exponent
  - If $m < n^{1/3}$, then $c = m^3 < n$ and can just take the cube root of $c$ to recover $m$
    - Even problems if “pad” $m$ in some ways [Hastad]
- Let $c_i = m^3 \mod n_i$ - same message is encrypted to three people
  - Adversary can compute $m^3 \mod n_1n_2n_3$ (using Chinese Remainder Theorem)
  - Then take ordinary cube root to recover $m$
Integrity in RSA Encryption

- Plain RSA does not provide integrity
- Given encryptions of $m_1$ and $m_2$, attacker can create encryption of $m_1 \cdot m_2$
  - $(m_1^e) \cdot (m_2^e) \mod n = (m_1 \cdot m_2)^e \mod n$
  - Attacker can convert $m$ into $m^k$ without decrypting
    - $(m^e)^k \mod n = (m^k)^e \mod n$

- In practice, OAEP is used: instead of encrypting $M$, encrypt $M \oplus G(r) \oplus H(M \oplus G(r))$
  - $r$ is random and fresh, $G$ and $H$ are hash functions
  - Resulting encryption is plaintext-aware: infeasible to compute a valid encryption without knowing plaintext
    - … if hash functions are “good” and RSA problem is hard
OAEP Encryption

- To encode,
  - messages are padded with \( k1 \) zeros to be \( n − k0 \) bits in length
  - \( r \) is a random \( k0 \) bit string
  - \( G \) expands the \( k0 \) bits of \( r \) to \( n − k0 \) bits.
  - \( X = m00..0 \oplus G(r) \)
  - \( H \) reduces the \( n − k0 \) bits of \( X \) to \( k0 \) bits.
  - \( Y = r \oplus H(X) \)
  - The output is \( X || Y \) where \( X \) is shown in the diagram as the leftmost block and \( Y \) as the rightmost block.

- To decode,
  - recover random string as \( r = Y \oplus H(X) \)
  - recover message as \( m00..0 = X \oplus G(r) \)
Recent work (soon to be published) by Lenstra et al and Heninger et al shows many keys out there generated with bad randomness

~ 4% RSA moduli shared between two certificates

Most crypto libraries use /dev/urandom even though no guaranteed entropy (especially on boot)

Real, concrete problem in applied security
Digital Signatures

**Given:** Everybody knows Bob’s public key
Only Bob knows the corresponding private key

**Goal:** Bob sends a “digitally signed” message
1. To compute a signature, must know the private key
2. To verify a signature, enough to know the public key
Digital Signature Properties

• Authentication - “It’s really Bob that sent this”

• Nonrepudiation - “Bob can’t later claim he didn’t mean this”

• Integrity - “This is the thing Bob meant to send”
RSA Signatures

• Public key is (n,e), private key is d
• To sign message m: \( s = m^d \mod n \)
  • Signing and decryption are the same operation in RSA (not true for all schemes)
• It’s infeasible to compute s on m if you don’t know d
• To verify signature s on message m: \( s^e \mod n = (m^d)^e \mod n = m \)
  • Just like encryption
• Anyone who knows n and e (public key) can verify signatures produced with d (private key)
• In practice, also need padding & hashing (why?)
More on Signing

• Decryption not always signature
• Sign a hash not the message
• Signing a hash image with size equal to modulus is provably secure
Digital Signature Attacks

• Attack models (GMR)
  • key only (only public key)
  • known message (have some messages)
  • adaptive chosen message (can get chosen messages before attack)

• Attack Results
  • total break (recovery of signing key)
  • universal forgery (forge signatures in all messages)
  • selective forgery (adversary can create and sign some messages)
  • existential forgery (some valid but unchosen msg/signature pair created)

• Provably secure - No existential forgery under adaptive chosen message attack
Public Key Infrastructure (PKI)

- Only secure if binding between public key and owner is correct
- Approaches to verifying this
  - Hierarchical certificate authorities (x509)
  - Local trust model (SPKI/SDSI)
  - Web of trust (PGP/GPG)