CS 555: Cryptography

Constructing Secure Encryption Schemes against Eavesdropper
Constructing Secure Encryption Schemes

Building Blocks:

- Pseudorandom Generators
- Stream Ciphers
A PRG (call it G) uses a short random string s (called seed) to generate a long "random-looking" string G(s).

How to measure quality of G?

Old way: Use a set of statistical tests, in which a statistical test D would output 1 if G passes that particular test.

- No guarantee that G will not fail some future newly invented test.
Modern quality requirement for a PRG

- Applies to any efficient statistical test D, even future yet-to-be-invented tests:

For any efficient statistical test D (henceforth called a distinguisher), the probability that D returns 1 when given the output of the PRG should be close to the probability that D returns 1 when given a uniform string of the same length.
Security of a PRG

- **Input:** *Short* random seed \( s \in \{0,1\}^n \)
- **Output:** Longer “pseudorandom” string \( G(s) \in \{0,1\}^{\ell(n)} \) with \( \ell(n) > n \)
  - \( \ell(n) \) is called expansion factor
- **PRG Security:** For all PPT distinguishers \( D \) there is a negligible function \( \text{negl}(\cdot) \) s.t
  \[
  \left| \Pr_{s \in \{0,1\}^n}[D(G(s)) = 1] - \Pr_{R \in \{0,1\}^{\ell(n)}}[D(R) = 1] \right| \leq \text{negl}(n)
  \]
- **Concrete Security:** We say that \( G(.) \) is a \((t(n), \varepsilon(n))\)-secure PRG if for all attackers running in time at most \( t(n) \) we have
  \[
  \left| \Pr_{s \in \{0,1\}^n}[D(G(s)) = 1] - \Pr_{R \in \{0,1\}^{\ell(n)}}[D(R) = 1] \right| \leq \varepsilon(n)
  \]
PRG Security as Game against D

- Seed $s \in \{0,1\}^n$ is uniformly generated
- $s$ is used to obtain $m_1 = G(s)$
- $m_0 \in \{0,1\}^{\ell(n)}$ is uniformly generated
- Random bit $b$ is generated
- $m_b$ is used as input to $D$, returning bit $b'$
- If $b' = b$ then $D$ has won the game

Security: $\Pr(D \text{ wins}) \leq 0.5 + \operatorname{negl}(n)$
How far from uniform is G’s output?

- The output of G is far from being uniform
  - Even though it is indistinguishable from uniform for a PPT distinguisher D
- If, for example, $\ell(n) = 2n$, then at most $2^n$ of the $2^{2n}$ strings of length $2n$ can be output by G
  - This is only a tiny fraction (1 out of $2^n$) of the strings of length $2n$
- What if D is not PPT?
What if D is not limited to PPT?

- Suppose D can use exponential time computation in the previous example of $\ell(n) = 2n$
- Such a D can afford to try all values of s and return $b' = 1$ if $m_b$ is $G(s)$ for some s
  - If $b$ is 1 then it returns $b' = 1$ with probability 1
  - If $b$ is 0 then it returns $b' = 1$ with probability $2^{-n}$
  - The difference in probabilities is $\sim 1$ (not negligibly small)
- But a practical D has to be PPT (luckily)
One-Time-Pads + PRGs

- Encryption:
  - Secret key is the seed ($K=s$)
    $$\text{Enc}_s(m) = G(s) \oplus m$$
    $$\text{Dec}_s(c) = G(s) \oplus c$$
  - **Advantage**: $|m| = \ell(n) > |s| = n$
  - Computational Security vs perfect security
  - **Disadvantage**: Still can only send one message

**Theorem** If $G$ is a PRG then the above encryption scheme has indistinguishable encryptions in the presence of an eavesdropper
Stream Cipher

- PRG produces its output all at once, whereas a stream cipher outputs the bits as a stream (one bit at a time and “on demand”, which is more efficient if few bits are needed)

- PRG outputs a fixed number of bits, whereas a stream cipher is not limited to a fixed number of bits (has greater flexibility)
Stream Cipher

- Two deterministic algorithms
  - Init
  - GetBits
- Init
  - Input: seed $s$, initialization vector IV (optional)
  - Output: an initial state $s_{t_0}$
- GetBits
  - Input: a state $s_{t_i}$
  - Output: a bit $y$ and an updated state $s_{t_{i+1}}$
Stream Cipher

- Iterating a stream cipher \( \ell \) times gives an \( \ell \) bit output, defining a function \( G_\ell \):

\[
\begin{align*}
st_0 & := \text{Init}(s,IV) \\
\text{For } i=1 \text{ to } \ell: & \quad (y_i, st_i) := \text{GetBits}(st_{i-1}) \\
\text{Output stream: } y_1, \ldots, y_\ell
\end{align*}
\]

- The stream cipher is secure if it takes no IV and, for any polynomial \( \ell \), the above function \( G_\ell \) is a PRG of expansion factor \( \ell \)
The RC4 Stream Cipher

- A proprietary cipher owned by RSA, designed by Ron Rivest in 1987
- Simple and effective design.
- Variable key size (typical 40 to 256 bits)
- Output unbounded number of bytes
- Widely used (SSL/TLS, wireless WEP)
- **Newer Versions**: RC5 and RC6
The RC4 Cipher

- The cipher internal state consists of
  - a 256-byte array $S$ containing permutation of size 256 (number of possible states is $256! \approx 2^{1700}$)
  - two indexes: $i, j$

\[
i = j = 0
\]

Loop
\[
i = (i + 1) \pmod{256}
\]
\[
j = (j + S[i]) \pmod{256}
\]
\[
\text{swap}(S[i], S[j])
\]
\[
\text{output } S[S[i] + S[j]] \pmod{256}
\]
End Loop
Limitations of Security Def. Used so Far

- Assumes adversary observes one ciphertext, can be weak if adversary observes 2 of them
  - For example, if the 2 ciphertexts are
    \[ c_1 = \text{Enc}_s(m_1) = G(s) \oplus m_1 \]
    \[ c_2 = \text{Enc}_s(m_2) = G(s) \oplus m_2 \]
    then adversary can compute \( c_1 \oplus c_2 = m_1 \oplus m_2 \)
  - Doesn’t prevent adversary from modifying c, m
    - For example, if \( c = G(s) \oplus m \) an adversary A can flip a bit and change the amount in
      \[ m = \text{“Pay to A the amount (USD) of: 000000101”} \]
The multiple-message eavesdropping experiment \( \text{PrivK}^{\text{mult}}_{A,\Pi}(n) \):

1. The adversary \( A \) is given input \( 1^n \), and outputs a pair of equal-length lists of messages \( \vec{M}_0 = (m_{0,1}, \ldots, m_{0,t}) \) and \( \vec{M}_1 = (m_{1,1}, \ldots, m_{1,t}) \), with \( |m_{0,i}| = |m_{1,i}| \) for all \( i \).

2. A key \( k \) is generated by running \( \text{Gen}(1^n) \), and a uniform bit \( b \in \{0,1\} \) is chosen. For all \( i \), the ciphertext \( c_i \leftarrow \text{Enc}_k(m_{b,i}) \) is computed and the list \( \vec{C} = (c_1, \ldots, c_t) \) is given to \( A \).

3. \( A \) outputs a bit \( b' \).

4. The output of the experiment is defined to be 1 if \( b' = b \), and 0 otherwise.
Extend Definition to Multiple Messages

For scheme $\Pi = (Gen, Enc, Dec)$ define a random variable:

$$PrivK_{A,\Pi}^{mult}(1^n) = \begin{cases} 1 & \text{if } b = b' \\ 0 & \text{otherwise} \end{cases}$$

$\Pi$ has indistinguishable multiple encryptions in the presence of an eavesdropper if for all PPT adversaries $A$ there is a negligible function $\mu$ such that

$$\Pr[PrivK_{A,\Pi}^{mult}(1^n) = 1] \leq \frac{1}{2} + \mu(n)$$
Multiple vs Single Encryptions

If \( \Pi \) has *indistinguishable multiple encryptions* in the presence of an eavesdropper

then

\( \Pi \) also has *indistinguishable encryptions* in the presence of an eavesdropper

**Question:** Are the definitions equivalent?

**Answer:** No, *indistinguishable multiple encryptions* is a strictly stronger security notion
Example

\[ \text{Enc}_s(m) = G(s) \oplus m \]
\[ \text{Dec}_s(c) = G(s) \oplus c \]

Recall: \( \Pi = (Gen, Enc, Dec) \) has indistinguishable encryptions in the presence of an eavesdropper

Claim: \( \Pi = (Gen, Enc, Dec) \) does not have indistinguishable multiple encryptions in the presence of an eavesdropper
Multiple Message Eavesdropping Attack

- A outputs
  \[ m_0 = (0^{\ell(n)}, 0^{\ell(n)}) \]
  \[ m_1 = (0^{\ell(n)}, 1^{\ell(n)}) \]
- A receives back
  \[ c_1 = G(s) \oplus m_{b,1} \]
  \[ c_2 = G(s) \oplus m_{b,2} \]
- A chooses \( b' = 1 \) if \( c_1 \neq c_2 \), else \( b' = 0 \)

[ Attack works for any deterministic Enc ]
Did We Cheat?

- Attack specifically exploited the fact that A can ask to see multiple encryptions of the same message...
- The above argument might appear to show that no encryption scheme provides secure *indistinguishable multiple encryptions* in the presence of an eavesdropper

**Theorem**: If $\Pi$ is an encryption scheme where Enc is deterministic, then $\Pi$ does **not provide** secure *indistinguishable multiple encryptions*

Note: Implicit assumption that schemes are stateless
Where to go from here?

**Option 1:** Weaken the security definition so that adversary cannot request two encryptions of the same message.

- Undesirable, as data may contain repetitions, e.g., many people with the same last name
- We will actually want to strengthen the definition later…

**Option 2:** Consider *randomized* encryption algorithms