IERG4150
Intro. to Cryptography

Sherman Chow
Chinese University of Hong Kong
Fall 2022
Lecture 1b: One-Time Pad

Attacker's Goals against Encryption

- Recover the plaintext
- Recover a part of the plaintext
  - (Weaker adversary, weaker scheme may suffice)
- Recover the secret key
  - (Stronger)
- We will see more different goals later…
- Or will we [*]?

One-Time Pad (OTP) based on XOR

- XOR: For $b_1 \oplus b_2$ where b_i is a bit, see the table below
- For bit-string operation $S_1 \oplus S_2$, just $\oplus$ in a bit-wise manner

- OTP = {KeyGen, Enc, Dec}
- KeyGen(1^λ):
  - sample *k uniformly* from the set of λ-bit strings
  - output *k*
- Enc(*k*, *m*) → $c = m \oplus k$;
  - (*m* is λ-bit long)
- Dec(*k*, *c*) → $m = c \oplus k$

$$\frac{\textbf{KeyGen:}}{k \leftarrow \{0,1\}^\lambda \\ \textbf{return } k}$$

$$\frac{\text{Enc}(k, m \in \{0,1\}^\lambda):}{\textbf{return } k \oplus m}$$

$$\frac{\text{Dec}(k, c \in \{0,1\}^\lambda):}{\textbf{return } k \oplus c}$$

XOR ⊕	0	1
0	0	1
1	1	0

Example

- OTP-encrypt the 20-bit plaintext *m* below under the key *k*:

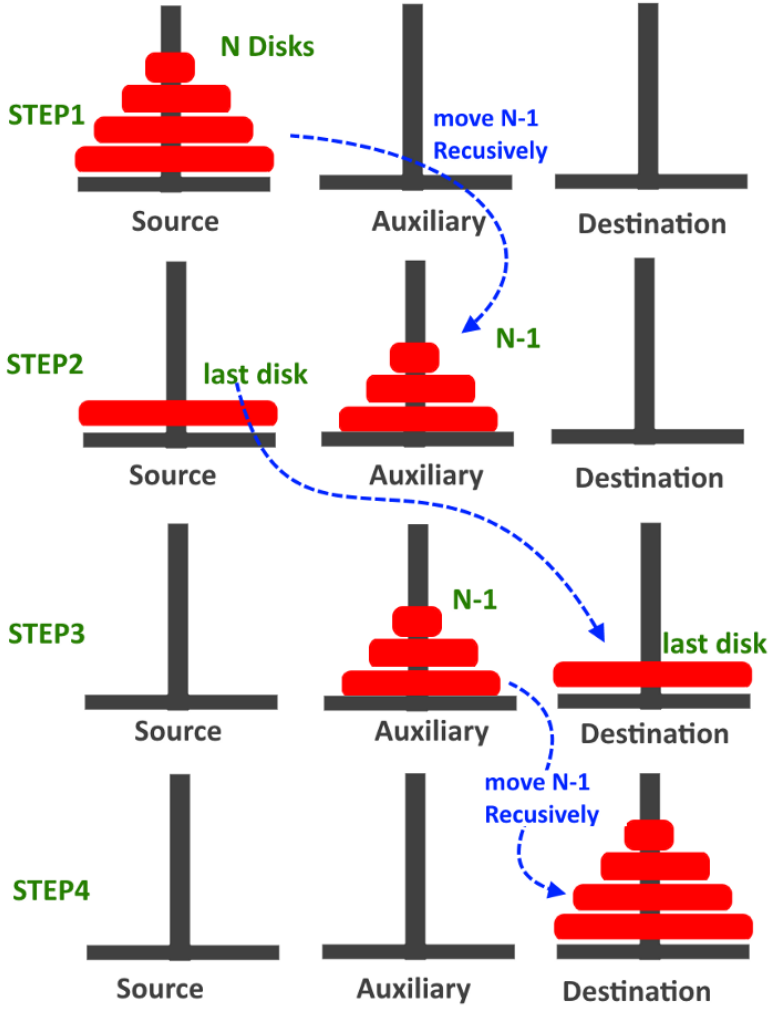
$$\begin{array}{lll} & 11101111101111100011 & (m) \\ \oplus & 00011001110000111101 & (k) \\ \hline & 11110110011111011110 & (c = \mathrm{Enc}(k, m)) \end{array}$$

- OTP-decrypt the 20-bit plaintext *m* below under the key *k*:

$$\begin{array}{lll} & 00001001011110010000 & (c) \\ \oplus & 10010011101011100010 & (k) \\ \hline & 10011010110101110010 & (m = \mathrm{Dec}(k, c)) \end{array}$$

Detour: Algorithms

- I could dry-run an algorithm with concrete examples if I were teaching an algorithm course
  - Not exactly on the right, "abstracted away" by the recursive call
  - but I could flatten it out if I want to



Credit:
https://medium.com/@jamalmaria111/tower-of-hanoi-js-algorithm-3f667fa46f0f

Crypto. Algorithms

- I have provided concrete examples, what did u learn?

$$
\begin{array}{lll}
 & 11101111101111100011 & (m) \\
\oplus & 00011001110000111101 & (k) \\
\hline
 & 11110110011111011110 & (c = \mathrm{Enc}(k, m))
\end{array}
$$

- You saw how Enc() (or Dec()) works for a particular input
- You get a *sense* of correctness (m = Dec(k, Enc(k, m)))
- But how can you argue about its security?

Why Cryptography is difficult?

- Security is a **global** property about the behavior of a system across **all possible** inputs.
  - You can't demonstrate security by example,
  - and there's nothing to see in a particular execution of an algorithm.
- Security is about a **higher level of abstraction**.
  - (and some students might not be comfortable with it)
- Most security definitions in this course are essentially:
  - "*the thing is secure if its outputs look like random junk.*"
  - *i.e.*, any example just look like meaningless garbage

Correctness of OTP

- For all $k, m \in \{0,1\}^\lambda$, it is true that $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$.
- More precisely: For all m in the *message space* $\mathbf{M} = \{0,1\}^\lambda$ and all k in the *key space* $\mathbf{K} = \{0,1\}^\lambda$, it is true that $\mathrm{Dec}(k, \mathrm{Enc}(k, m)) = m$.
- Or simply, one can *always* recover m.
- Proof:
- $\mathrm{Dec}(k, \mathrm{Enc}(k, m))$
- $= \mathrm{Dec}(k, k \oplus m)$
- $= k \oplus (k \oplus m)$
- $= (k \oplus k) \oplus m$ // $\oplus$ is associative: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$
- $= 0^\lambda \oplus m = m$.

XOR ⊕	0	1
0	0	1
1	1	0

Cautions

- (patented in 1919, but recently discovered in an 1882 text)
- The security crucially depends on sampling *k uniformly* at random from the set of λ-bit strings
  - The security would not hold if it is under other distribution.
- (This step in) KeyGen() is the only source of randomness
  - we'll see algorithms using "randomness "more" later
- Enc and Dec are essentially the same algorithm
  - but it is more of a coincidence than something truly fundamental
- Message space, key space, are just the ciphertext space
  - again, just a special case here, nothing is said in general

Security Proof

- "*Because of the specific way the ciphertext was generated, it doesn't reveal any information about the plaintext to the attacker, no matter what the attacker does with the ciphertext.*"

- We need to first specify how the ciphertext is generated.
- Didn't we? It is the encryption algorithm
  - (which relies on KeyGen())
- But it was from the point of view of "honest" users Alice and Bob
- But "no matter what the attacker does with the ciphertext"?
  - Yes, but at least we need to specify what ciphertext does *it* see.

Modelling what the adversary sees

- We always treat the attacker as some (unspecified) process that receives output from an algorithm (EAVESDROP here).

- not what the attacker does
- but rather the process
  (carried out by honest users)
  that produces what the attacker sees

$\underline{\textsc{Eavesdrop}(m \in \{0,1\}^\lambda):}$
$k \leftarrow \{0,1\}^\lambda$
$c := k \oplus m$
return c

- Our goal is to say something like
  "*the output of eavesdrop doesn't reveal the input m.*"

Probabilistic Alg. & its Output Distribution

- If you call eavesdrop several times,
- even on the same input,
- you are likely to get different outputs.

$\textbf{EAVESDROP}(m \in \{0,1\}^\lambda)$:
$k \leftarrow \{0,1\}^\lambda$
$c := k \oplus m$
return c

- Instead of thinking of "`eavesdrop`(*m*)" as a single string,
- think of it as a *probability distribution* over strings.
- Each time you call `eavesdrop`(*m*),
- you see a *sample* from that distribution.

(Toy) Example

- $\lambda = 3$ and consider `eavesdrop`(010) and `eavesdrop`(111).

EAVESDROP(010):

Pr	k	*output* $c = k \oplus 010$
1/8	000	010
1/8	001	011
1/8	010	000
1/8	011	001
1/8	100	110
1/8	101	111
1/8	110	100
1/8	111	101

EAVESDROP(111):

Pr	k	*output* $c = k \oplus 111$
1/8	000	111
1/8	001	110
1/8	010	101
1/8	011	100
1/8	100	011
1/8	101	010
1/8	110	001
1/8	111	000

every string in the ciphertext space $\{0, 1\}^\lambda$ appears exactly once, with the same (1/8) probability

a. k. a. uniform distribution over $\{0, 1\}^\lambda$

k is chosen uniformly at random from $\{0, 1\}^\lambda$

Some conclusions

- Nothing special about 010 or 111 in the above examples.
- The distribution `eavesdrop`(m) is the uniform distribution over the ciphertext space $\{0, 1\}^\lambda$.
- Let's formalize this argument (without tabulating 2^3 times).

- Let's first formalize what we want to prove:
- "For every $m \in \{0, 1\}^\lambda$, the distribution `eavesdrop`(m) is the uniform distribution on $\{0, 1\}^\lambda$."
- Corollary: For every $m, m' \in \{0, 1\}^\lambda$, the distributions `eavesdrop`($m$) and `eavesdrop`($m'$) are identical.

The Exact Proof from the Textbook

Proof Arbitrarily fix $m, c \in \{0,1\}^\lambda$. We will calculate the probability that $\textsc{eavesdrop}(m)$ produces output c. That event happens only when

$$c = k \oplus m \iff k = m \oplus c.$$

The equivalence follows from the properties of $\textsc{xor}$ given in Section 0.3. That is,

$$\Pr[\textsc{eavesdrop}(m) = c] = \Pr[k = m \oplus c],$$

where the probability is over uniform choice of $k \leftarrow \{0,1\}^\lambda$.

We are considering a specific choice for m and c, so there is *only one* value of k that makes $k = m \oplus c$ true (causes m to encrypt to c), and that value is exactly $m \oplus c$. Since k is chosen *uniformly* from $\{0,1\}^\lambda$, the probability of choosing the particular value $k = m \oplus c$ is $1/2^\lambda$.

In summary, for every m and c, the probability that $\textsc{eavesdrop}(m)$ outputs c is exactly $1/2^\lambda$. This means that the output of $\textsc{eavesdrop}(m)$, for any m, follows the uniform distribution. ■

Why did we prove? (Part I)

- "For every $m \in \{0,1\}^\lambda$, the distribution `eavesdrop`(m) is the uniform distribution on $\{0,1\}^\lambda$"; or (in "English"):
  - *"If an attacker sees a single ciphertext,*
  - *encrypted with one-time pad, where the key*
  - *is chosen uniformly and kept secret from the attacker,*
  - *then the ciphertext appears uniformly distributed."*
- Suppose someone chooses a plaintext *m*.
- You (the attacker) get to see the resulting ciphertext —
- a sample from the distribution you can sample by yourself
- even if you don't know *m*!

Security of OTP, and some discussions

- The "real" ciphertext doesn't carry *any information* about *m* if it is possible to sample without even knowing *m*!

- Paradox 1: "One can *always* recover *m* [from c]" contradicts with "c contains no information about *m*."
- The correctness proof assumes one w/ the knowledge of *k*
- Paradox 2: "`eavesdrop`(*m*) does not depend on *m*" is blatantly false simply because it takes *m* as an input!
- Our example shows that, when *m* is different,
  the tabulated outputs indeed are different (*m*'s "effect")
- The claim is about they are being the same distribution.

Why did we prove? (Part II)

- For every m, $m' \in \{0, 1\}^{\lambda}$, the distributions `eavesdrop`($m$) and `eavesdrop`($m'$) are identical.
  - *"If an attacker sees a single ciphertext,*
  - *encrypted with one-time pad, where the key*
  - *is chosen uniformly and kept secret from the attacker,*
  - *for every two possibilities of the plaintext,*
  - *the resulting ciphertext appears from the same distribution"*
- The attacker's "view" is the same no matter what m is
- and no matter what the plaintext distribution is!
  - (*cf.*, Caesar cipher…)

What did we prove? (Part III)

- "For every $m \in \{0, 1\}^\lambda$, the distribution `eavesdrop`(m) is the uniform distribution on $\{0, 1\}^\lambda$"

- Here, we consider some hypothetical "ideal" world:
- Any attacker sees only a source of uniform bits.
- There are no keys and no plaintexts to recover.

What did we prove? (fin.)

- "For every $m \in \{0, 1\}^\lambda$, the distribution `eavesdrop`(m) is the uniform distribution on $\{0, 1\}^\lambda$"

- Nothing was said about the attacker's goal!
  - *e.g.*, recovering the plaintext or the key
  - Looking ahead, we may do that in alternative definitions or cases
  - but we still want to be general enough
- What we prove: Any attacker, who saw an OTP ciphertext in the real world, has a point of view like in our hypothetical world!
- Or, it is a "modest" goal: detect that ciphertexts don't follow a uniform distribution (so harder goals are out of reach)

Limitations of One-Time Pad

1. It can only be used once (to encrypt a single plaintext).
   - Note that the eavesdrop procedure provides no way for a caller to guarantee that two calls will use the same key.
   - So, we did not prove anything about reusing the key.
2. The key is as long as the plaintext
   - provably unavoidable (a.k.a. the key length is optimal) [*]
   - Chicken-and-egg dilemma in practice:
     - If two users want to privately convey a λ-bit message,
     - they first need to privately agree on a λ-bit string.

Then why teach OTP?

- Practical: It is the only encryption scheme that is "perfectly secure," so you know what to do if someone is selling a "perfect" encryption scheme to you…
- Pedagogical: It illustrates fundamental ideas that appear in most forms of encryption in this course.

- We propose the first solution, then we try to "twist" it to make it achieve some "better trade-offs"
  - How "innovation" work sometimes
- What if the attacker has bounded computation power?
- What if we manage to have some "pseudorandom strings"?

Shannon's Information-Theoretic Security [**]

- "A priori probability of a plaintext message *m* is the same as the a posteriori probability of *m* given the corresponding ciphertext *c*."
- H(x) --- A measure of the amount of information (about x)
  - that is missing before reception
- Consider a fair coin, $Pr(x_0) = Pr(x_1) = ½$
- $- \lg(1/2) = -(\lg 1 - \lg 2) = -(0 - 1) = 1$
  - Here, lg means $\log_2$ (so lg 2 = 1), not following "ISO notation"…
- So, we use 1 bit to denote this event (of coin tossing)
  - with probability ½ we gain this 1 bit of info is 0 (or 1)
- $H(m) = H(m \mid c)$
  - R.H.S.: conditional entropy of the plaintext given the ciphertext
- This is a definition of confidentiality

$$H(x) = -\sum_{i=0}^{n-1} \Pr(x_i) \log \Pr(x_i)$$

OTP is Information-Theoretically Secure [*]

- $\Pr(M = m \mid C = c)$

  $= \Pr(M = m \wedge C = c) / \Pr(C = c)$ [Bayes' law]

  $= \Pr(M = m) \Pr(C = c \mid M = m) / \Pr(C = c)$ [prob. def.]

  $= \Pr(M = m) \Pr(C = c \mid M = m)$

  $/ \Sigma_{m' \in M} \Pr(M = m') \Pr(C = c \mid M = m')$

  $= \Pr(M = m) \cdot (1/2^{\lambda}) / \Sigma_{m' \in M} \{ \Pr(M = m') \cdot (1/2^{\lambda}) \}$

  $= \Pr(M = m) / 1$

  since $\Pr(C = c \mid M = m') = 1/2^{\lambda}$ for all c and m' for OTP

Spoiler: "Compressed" Secret-Keys

- Pseudo-random number generator (PRNG)
  - outputs a long string of "random-looking" bits
  - from a short random seed
  - a.k.a. stream cipher
- Computationally secure against the "Next-bit test"
  - given the first k bits of a random sequence
  - no polynomial-time algorithm can predict the $(k+1)^{th}$ bit
  - with probability of success better than 50%
  - a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness [Yao82]