Introduction to Modern Cryptography

Michele Ciampi

(Slides courtesy of Prof. Jonathan Katz)

Lecture 5, part 1

So far

- Introduced **perfect secrecy** (PS)
- Introduced **OTP** and proved that it satisfies PS
- Described the two **limitations of the OTP**
  1. Key as long as the message
  2. Key used only once
- Introduced **perfect indistinguishability** (PI)
- Proved that **PI is equivalent to PS**

This lecture

- Relax PI to **computational secrecy** (CS): a weaker, yet practical notion of security
- Introduce **pseudorandom generators** (PRG)

Computational Secrecy

Computational Secrecy?

Idea

Relax perfect indistinguishability

Two approaches

- Concrete security
- Asymptotic security

Computational Indistinguishability (Concrete)

Concrete Approach

- $(\boldsymbol{t}, \boldsymbol{\epsilon})$-indistinguishability:
- Security may fail with probability $\leq \boldsymbol{\epsilon}$
- Restrict attention to attackers running in time $\leq \boldsymbol{t}$
  - Or in $\boldsymbol{t}$ CPU cycles

Computational Indistinguishability (Concrete)

Concrete Approach

$\mathbf{\Pi}$ is $\boldsymbol{(t, \epsilon)}$-indistinguishable if for all attackers $\boldsymbol{A}$ running in time at most $\boldsymbol{t}$, it holds that

$$\mathbf{Pr}[\mathsf{PrivK}_{\boldsymbol{A},\mathbf{\Pi}} = \mathbf{1}] \leq \frac{\mathbf{1}}{\mathbf{2}} + \boldsymbol{\epsilon}$$

Note

- $\boldsymbol{(\infty, 0)}$-indistinguishable = perfect indistinguishability
- Relax definition by taking $\boldsymbol{t} < \boldsymbol{\infty}$ and $\boldsymbol{\epsilon} > \mathbf{0}$

Concrete Security

Drawbacks

- Parameters $\boldsymbol{t, \epsilon}$ are what we ultimately care about in the real world
- Does not lead to a clean theory:
  - Sensitive to exact computational model
  - $\boldsymbol{\Pi}$ can be $\boldsymbol{(t, \epsilon)}$-secure for many choices of $\boldsymbol{t, \epsilon}$
- Would like to have schemes where users can adjust the achieved security as desired

Asymptotic Security

- Introduce security parameter $\boldsymbol{n}$
  - e.g. think of $\boldsymbol{n}$ as the key length
  - Chosen by honest parties when they generate/share key
  - Allows users to tailor the security level
  - Known by adversary
- Measure running times of all parties, and the success probability of the adversary, as functions of $\boldsymbol{n}$

Computational Indistinguishability (Asymptotic)

Asymptotic Approach

- Security may fail with probability **negligible** in $\boldsymbol{n}$
- Restrict attention to attackers running in time (at most) **polynomial** in $\boldsymbol{n}$

Polynomial Function

$\boldsymbol{Z^+ = \{1, 2, 3, \ldots\}}$ – set of positive integers

Definition

A function $\boldsymbol{f : Z^+ \to Z^+}$ is **polynomial** if there exists $\boldsymbol{c}$ such that $\boldsymbol{f(n) < n^c}$.

i.e. $\boldsymbol{f}$ is **asymptotically bounded** by a polynomial

Notation

poly$\boldsymbol{(n)}$ or just **poly** – any polynomial function in $\boldsymbol{n}$

Negligible Function

Definition

A function $f : Z^+ \to [0,1]$ is **negligible** if for every polynomial p, $\exists N$ s.t. $\forall n > N :\ f(n) < \frac{1}{p(n)}$.

i.e. f **decays faster** than any inverse poly. for *large enough* n

Definition (equivalent)

A function $f : Z^+ \to [0,1]$ is **negligible** if $\forall c = \mathbf{const} :\ \exists N$ s.t. $\forall n > N :\ f(n) < n^{-c}$.

Notation

$\mathbf{negl}(n)$ or just **negl** – any negligible function in n

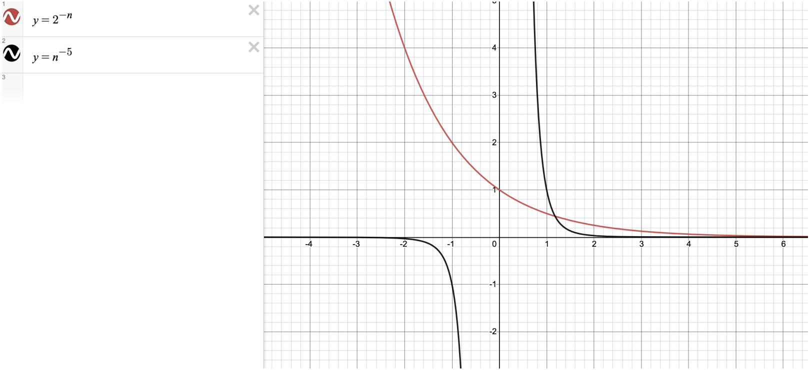
Examples of Negligible Functions

Let $\boldsymbol{p(n) = n^{-5}}$ i.e. $\boldsymbol{c = 5}$.

$\boldsymbol{f(n) = 2^{-n}}$

Solve $\boldsymbol{2^{-n} < n^{-5} \implies n > 5 \log n}$ for $\boldsymbol{n \geq 23}$ i.e. $\boldsymbol{N = 22}$

n^{-5} vs 2^{-n}



Examples of Negligible Functions

Let $\boldsymbol{p(n) = n^{-5}}$ i.e. $\boldsymbol{c = 5}$.

$\boldsymbol{f(n) = 2^{-n}}$

Solve $\boldsymbol{2^{-n} < n^{-5} \implies n > 5 \log n}$ for $\boldsymbol{n \geq 23}$ i.e. $\boldsymbol{N = 22}$

Solve $\boldsymbol{2^{-\sqrt{n}} < n^{-5} \implies n > 25 \log^2 n}$ for $\boldsymbol{n \geq\approx 3500}$

$\boldsymbol{f(n) = n^{-\log n}}$

Solve $\boldsymbol{n^{-\log n} < n^{-5} \implies \log n > 5}$ for $\boldsymbol{n \geq 33}$

i.e. $\forall \boldsymbol{c}$, $\boldsymbol{f(n)}$ decays faster than $\boldsymbol{n^{-c}}$ for large enough $\boldsymbol{n}$.

Examples of Negligible Functions

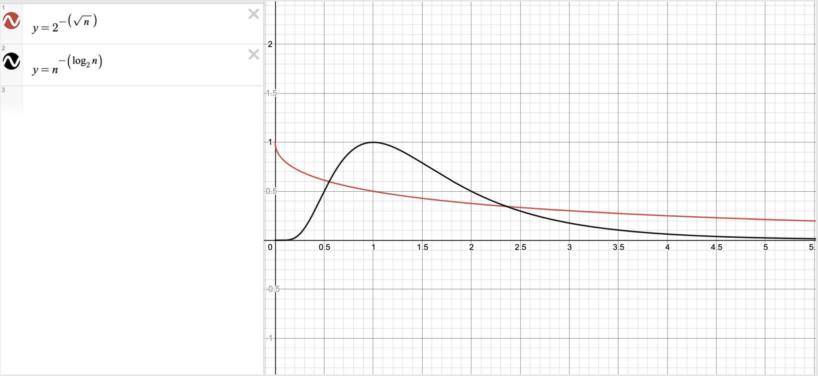
Warning!

- **Wrong:** $\boldsymbol{n^{-\log n}}$ decays faster than $\boldsymbol{2^{-\sqrt{n}}}$
- Note: $\boldsymbol{2^{-\sqrt{n}} < n^{-\log n} : \forall n > 65536}$.
- **Correct:** $\boldsymbol{n^{-\log n}}$ decays faster than $\boldsymbol{2^{-\sqrt{n}}}$ for $\boldsymbol{n \leq 65536}$

For values $\boldsymbol{n < 65536}$ an adversarial sucess probability of $\boldsymbol{n^{-\log n}}$ is still preferrable (for the algorithm designer) to $\boldsymbol{2^{-\sqrt{n}}}$

Examples of Negligible Functions

Warning!



For values $\boldsymbol{n < 65536}$ an adversarial sucess probability of $\boldsymbol{n^{-\log n}}$ is still preferrable (for the algorithm designer) to $\boldsymbol{2^{-\sqrt{n}}}$

Why polynomial? Why negligible?

- Somewhat arbitrary choices
- Borrowed from complexity theory
- **efficient = probabilistic polynomial-time (PPT)**
- Convenient closure properties

Closure Properties

- **poly · poly = poly**
  - A PPT algorithm making calls to PPT subroutines is PPT
- **poly · negl = negl**
  - Poly-many calls to subroutines that fail with negligible probability fail with negligible probability overall

Redefining Encryption in the Computational Setting

A private-key encryption scheme is defined by three **PPT** algorithms (Gen,Enc,Dec) :

- Gen: takes as input $\mathbf{1^n}$; outputs $\boldsymbol{k}$. (Assume $|\boldsymbol{k}| \geq \boldsymbol{n}$.)
- Enc: takes as input a key $\boldsymbol{k}$ and message $\boldsymbol{m} \in \{\mathbf{0}, \mathbf{1}\}^*$; outputs ciphertext $\boldsymbol{c} \leftarrow \mathsf{Enc}_{\boldsymbol{k}}(\boldsymbol{m})$
- Dec: takes key $\boldsymbol{k}$ and ciphertext $\boldsymbol{c}$ as input; outputs a message $\boldsymbol{m}$ or **error**

The $\mathbf{1}^{\boldsymbol{n}}$ notation

$$\mathbf{1}^{\boldsymbol{n}} = \underbrace{\mathbf{11\ldots1}}_{\boldsymbol{n}\ \textbf{times}}$$

- Denotes the size of the input e.g. $\mathsf{Gen}(\mathbf{1}^{\boldsymbol{n}})$ or $\boldsymbol{A}(\mathbf{1}^{\boldsymbol{n}})$
- Stresses that a PPT algorithm (e.g. Gen, $\boldsymbol{A}$) is **polynomial in $\boldsymbol{n}$**

Computational Indistinguishability (Asymptotic)

$\mathsf{PrivK}_{A,\Pi}(n)$

Fix a scheme Π and some adversary A. Define a randomized experiment $\mathsf{PrivK}_{A,\Pi}(n)$:

- $A(1^n)$ outputs $m_0, m_1 \in \{0,1\}^*$ of equal length
- $k \leftarrow \mathsf{Gen}(1^n)$, $b \leftarrow \{0,1\}$, $c \leftarrow \mathsf{Enc}_k(m_b)$
- $b' \leftarrow A(c)$
- Adversary A succeeds if $b = b'$, and we say the experiment evaluates to 1 in this case

Computational Indistinguishability (Asymptotic)

$\mathbf{\Pi}$ is **computationally indistinguishable (EAV-secure)** if for all PPT attackers $\boldsymbol{A}$, there is a **negligible function** $\boldsymbol{\epsilon}$ such that

$$\mathbf{Pr}[\mathsf{PrivK}_{\boldsymbol{A},\mathbf{\Pi}}(\boldsymbol{n}) = \mathbf{1}] \leq \frac{\mathbf{1}}{\mathbf{2}} + \boldsymbol{\epsilon}(\boldsymbol{n})$$

EAV-secure = indistinguishable against EAVesdropping

- Note that $\boldsymbol{f}(\boldsymbol{n}) = \mathbf{Pr}[\mathsf{PrivK}_{\boldsymbol{A},\mathbf{\Pi}}(\boldsymbol{n}) = \mathbf{1}]$ is a function in $\boldsymbol{n}$
- $\boldsymbol{f} : \boldsymbol{Z}^{+} \rightarrow [\mathbf{0}, \mathbf{1}]$ maps each value of $\boldsymbol{n}$ to a probability
- Therefore we can talk about the **asymptotic behaviour** of $\boldsymbol{f}$ in the security parameter $\boldsymbol{n}$

Example (EAV-security)

Consider a scheme $\mathbf{\Pi}$ where $\mathsf{Gen}(1^n)$ generates a uniform n-bit key. Assume that we know that the best attack is brute-force search of the key space

Example (EAV-security)

$\boldsymbol{A}$'s attack strategy

1. Input $\boldsymbol{m_0, m_1, c}$; find $\boldsymbol{b : \mathsf{Enc}_k(m_b) = c}$.
2. $\boldsymbol{A}$ randomly selects $\boldsymbol{k \in \mathcal{K}}$ and computes $\boldsymbol{\mathsf{Enc}_k(m_0)}$ and $\boldsymbol{\mathsf{Enc}_k(m_1)}$.
3. If $\boldsymbol{k}$ is correct ($\boldsymbol{c}$ matches $\mathsf{Enc}_{\boldsymbol{k}}$) – output correct $\boldsymbol{b}$
4. Else output random guess $\boldsymbol{b}$
5. $\mathbf{Pr}$ of $\boldsymbol{A}$ to succeed i.e. $\mathbf{Pr}[\mathsf{PrivK}_{\boldsymbol{A},\boldsymbol{\Pi}}\boldsymbol{(n) = 1}]$ is:

$$\mathbf{Pr[correct\ guess|correct\ key]Pr[correct\ key]+}$$
$$\mathbf{Pr[correct\ guess|incorrect\ key]Pr[incorrect\ key]}$$
$$\boldsymbol{= 1\frac{1}{2^n} + \frac{1}{2}(1 - \frac{1}{2^n}) = \frac{1}{2} + \frac{1}{2^{n+1}} = \frac{1}{2} + \mathbf{negl}}$$

$\Longrightarrow$ $\boldsymbol{\Pi}$ is EAV-secure

Example (EAV-security)

Give more computational power to the attacker and assume $\boldsymbol{A}$ can make not **1** but $\boldsymbol{t(n)}$ key guess where $\boldsymbol{t}$ is **polynomial** in $\boldsymbol{n}$

Example (EAV-security)

A's attack strategy (polynomial adversary)

1. Input m_0, m_1, c; find $b : \mathsf{Enc}_k(m_b) = c$.
2. A randomly selects $t(n)$ keys $k \in \mathcal{K}$ and for each key computes $\mathsf{Enc}_k(m_0)$ and $\mathsf{Enc}_k(m_1)$.
3. If one k is correct (c matches Enc_k) – output correct b
4. Else output random guess b
5. $\mathbf{Pr}$ of A to succeed i.e. $\mathbf{Pr}[\mathsf{PrivK}_{A,\Pi}(n) = 1]$ is:

$$\frac{t(n)}{2^n} 1 + \left(1 - \frac{t(n)}{2^n}\right) \frac{1}{2} = \frac{1}{2} + \frac{t(n)}{2^{n+1}} = \frac{1}{2} + \mathbf{negl}$$

$\implies$ $\mathbf{\Pi}$ is EAV-secure

For polynomial t, the function $\frac{t(n)}{2^{n+1}}$ is **negligible**

- Recall: $\mathbf{poly} \cdot \mathbf{negl} = \mathbf{negl}$

Example

- What happens when computers get faster?
- e.g. consider a scheme that takes time n^2 to run but time 2^n to break with prob. 1
- What if computers get 4 times faster?
- Honest users double n and can thus maintain the same running time: $(2n)^2/4 = n^2$
- Time to break scheme is squared: 2^{2n}
  - Time required to break the scheme increases
- The security proofs still hold

Encryption and Plaintext Length

- In practice, we want encryption schemes that can encrypt arbitrary-length messages
- Encryption does not hide the plaintext length (in general)
- The definition takes this into account by requiring $\boldsymbol{m_0, m_1}$ to have the same length
- Beware that leaking plaintext length can often lead to problems in the real world
  - e.g. plaintexts *(yes, no)* or numerical values
  - e.g. compression before encryption: small length $\implies$ big plaintext redundancy (CRIME attack on TLS)

If leaking plaintext length is a concern, additional steps are necessary e.g. pad all messages to the same length.

Computational Secrecy

- From now on, we will assume the **computational setting** by default
- Usually, the **asymptotic setting**

End

References: Chapter 3, until Pag. 56.