INTRODUCTION TO PROVABLE SECURITY
Models, Adversaries, Reductions
CRYPTOGRAPHY / CRYPTOLOGY

- “from Greek κρυπτός kryptós, "hidden, secret"; and γράφειν graphein, "writing", or -λογία -logia, "study", respectively”

- “is the practice and study of techniques for secure communication in the presence of third parties (called adversaries).”

Source: www.wikipedia.org
Some cryptographic goals

- **Confidentiality**
  - Content of conversation remains hidden

- **Authenticity**
  - Message is really sent by specific sender

- **Integrity**
  - Message has not been modified

- **Privacy:**
  - Sensitive (user) data remains hidden

- **Covertcy**
  - The fact that a conversation is taking place is hidden

- ....
CONFIDENTIALITY

- Parties exchange messages
- Parties store documents (or strings e.g. passwords)

No unauthorized party can learn anything about contents.
AUTHENTICITY

- “Online”: Alice proves legitimacy to Bob in real-time fashion (interactively)
  
  No unauthorized party can impersonate a user

- “Offline”: Alice generates proof of identity to be verified offline by Bob
  
  No unauthorized party can forge the proof
INTEGRITY

- Parties send or receive messages

No modification to content of message(s)
HOW CRYPTOGRAPHY WORKS

- Use building blocks (primitives)
  - ... either by themselves (hashing for integrity)
  - ... or in larger constructions (protocols, schemes)

- Security must be guaranteed even if mechanism (primitive, protocol) is known to adversaries

- Steganography vs. cryptography:
  - Steganography: hide secret information in plain sight
  - Cryptography: change secret information to something else, then send it
A brief history

- “Stone age”: secrecy of algorithm
  - Substitution and permutation (solvable by hand)
    - Caesar cipher, Vigenère cipher, etc.

- “Industrial Age”: automation of cryptology
  - Cryptographic machines like Enigma
  - Fast, automated permutations (need machines to solve)

- “Contemporary Age”: provable security
  - Starting from assumptions (e.g. a one-way function), I build a scheme, which is “provably” secure in model
PART II
THE PROVABLE SECURITY METHOD
SECURITY BY TRIAL-AND-ERROR

- Identify goal (e.g. confidentiality in P2P networks)
- Design solution – the strategy:
  - Propose protocol
  - Search for an attack
  - If attack found, fix (go to first step)
  - After many iterations or some time, halt
- Output: resulting scheme

- Problems:
  - What is “many” iterations/ “some” time?
  - Some schemes take time to break: MD5, RC4...
**Provable Security**

- Identify goal. Define security:
  - Syntax of the primitive: e.g. algorithms (KGen, Sign, Vf)
  - Adversary (e.g. can get signatures for arbitrary msgs.)
  - Security conditions (e.g. adv. can’t sign fresh message)

- Propose a scheme (instantiate syntax)

- Define/choose security assumptions
  - Properties of primitives / number theoretical problems

- Prove security – 2 step algorithm:
  - Assume we can break security of scheme (adv. $\mathcal{A}$)
  - Then build “Reduction” (adv. $\mathcal{B}$) breaking assumption
THE ESSENCE OF PROVABLE SECURITY

- Core question: what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”

- Say a scheme is secure against all known attacks
  - ... will it be secure against a new, yet unknown attack?

- Step 1: Define your primitive (syntax)

Signature Scheme: algorithms (KGen, Sign, Vf)

* KGen(1^γ) outputs (sk, pk)
* Sign(sk, m) outputs S (prob.)
* Vf(pk, m, S) outputs 0 or 1 (det.)
THE ESSENCE OF PROVABLE SECURITY

- Core question: what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”

- Say a scheme is secure against all known attacks
  - ... will it be secure against a new, yet unknown attack?

- Step 2: Define your adversary

Adversaries \( \mathcal{A} \) can:
- know public information: \( \gamma, \) pk
- get no message/signature pair
- get list of message/signature pairs
- submit arbitrary message to sign
THE ESSENCE OF PROVABLE SECURITY

- Core question: what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”
- Say a scheme is secure against all known attacks
  - ... will it be secure against a new, yet unknown attack?
- Step 3: Define the security condition

Adversary \( \mathcal{A} \) can output fresh \((m,S)\) which verifies, with non-negligible probability (as a function of \( \gamma \))
THE ESSENCE OF PROVABLE SECURITY

- Core question: what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”

- Say a scheme is secure against all known attacks
  - ... will it be secure against a new, yet unknown attack?

- Step 4: Propose a protocol

  Instantiate the syntax given in Step 1.
  E.g. give specific algorithms for KGen, Sign, Vf.
THE ESSENCE OF PROVABLE SECURITY

- Core question: what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”

- Say a scheme is secure against all known attacks
  - ... will it be secure against a new, yet unknown attack?

- Step 5: Choose security assumptions

For each primitive in the protocol, choose assumptions

- Security Assumptions (e.g. IND-CCA encryption)
- Number Theoretical Assumptions (e.g. DDH, RSA)
**The Essence of Provable Security**

- **Core question:** what does “secure” mean?
  - “Secure encryption” vs. “Secure signature scheme”

- **Say a scheme is secure against all known attacks**
  - … will it be secure against a new, yet unknown attack?

- **Step 6: Prove security**

For each property you defined in steps 1-3:
- Assume there exists an adversary $A$ breaking that security property with some probability $\varepsilon$
- Construct reduction $B$ breaking some assumption with probability $f(\varepsilon)$
HOW REDUCTIONS WORK

- Security assumptions are baseline

- Reasoning:
  - If our protocol/primitive is insecure, then the assumption is broken
  - But the assumption holds (by definition)

- Conclusion: The protocol cannot be insecure

- Caveat:
  - Say an assumption is broken (e.g. DDH easy to solve)
  - What does that say about our protocol?

We don’t know!
PART III
ASSUMPTIONS
We need computational assumptions

- Take our signature schemes \((KGen, \text{Sign}, Vf)\)

- Correctness: if parameters are well generated, well-signed signatures always verify.
WE NEED COMPUTATIONAL ASSUMPTIONS

- Take our signature schemes (KGen, Sign, Vf)

- Unforgeability: no adversary can produce signature for a fresh message $m^*$
  
  But any $A$ can guess $sk$ with probability $\frac{1}{2^{|sk|}}$
WE NEED COMPUTATIONAL ASSUMPTIONS

Take our signature schemes (KGen, Sign, Vf)

- Unforgeability: no adversary can produce signature for a fresh message $m^*$
  
  And any $A$ can guess valid $\sigma$ with probability $\frac{1}{2|\sigma|}$
SOME COMPUTATIONAL ASSUMPTIONS

- Of the type: It is “hard” to compute $x$ starting from $y$.

- How hard?
  - Usually no proof that the assumption holds
  - Mostly measured with respect to “best attack”
  - Sometimes average-case, sometimes worst-case

- Relation to other assumptions:
  - $A_1 \rightarrow A_2$: break $A_2$ => break $A_1$ [stronger]
  - $A_1 \leftarrow A_2$: break $A_1$ => break $A_2$ [weaker]
  - $A_1 \Leftrightarrow A_2$: both conditions hold [equivalent]
Examples: DLog, CDH, DDH

- Background:
  - Finite field $F$, e.g. $\mathbb{Z}_p^*$ = {1, 2, ..., p-1} for prime p
  - Multiplication, e.g. modulo p: $2(p - 2) = 2p - 4 = p - 4$
  - Element $g$ of prime order $q | (p - 1)$:
    
    \[ g^q = 1 \pmod{p} \text{ AND } g^m \neq 1 \pmod{p} \forall m < q \]
  - Cyclic group $G = < g > = \{1, g, g^2, ..., g^{q-1}\}$

- DLog problem:
  - Pick $x \in_R \{1, ..., q\}$. Compute $X = g^x \pmod{p}$.
  - Given $(p, q, g, X)$ find $x$.
  - Assumed hard.
EXAMPLES: DLog, CDH, DDH

- **DLog problem:**
  - Pick $x \in_R \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$.
  - Given $(p, q, g, X)$ find $x$.
  - Assumed hard.
EXAMPLES: DLog, CDH, DDH

DLog problem:
- Pick \( x \in_R \{1, \ldots, q\} \). Compute \( X = g^x \pmod{p} \).
- Given \((p, q, g, X)\) find \( x \).
- Assumed hard.

CDH problem:
- Pick \( x, y \in_R \{1, \ldots, q\} \). Compute \( X = g^x \pmod{p} \);
  \( Y = g^y \pmod{p} \).
- Given \((p, q, g, X, Y)\) find \( g^{xy} \).

Just to remind you: \( g^{xy} = X^y = Y^x \neq XY = g^{x+y} \)

- Solve D-LOG \(\rightarrow\) Solve CDH
- Solve CDH \(\Leftrightarrow\) Solve D-LOG
Examples: DLog, CDH, DDH

- **DLog problem:**
  - Pick $x \in \mathbb{R} \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$.
  - Given $(p, q, g, X)$ find $x$.

- **CDH problem:**
  - Pick $x, y \in \mathbb{R} \{1, \ldots, q\}$. Compute $X = g^x \pmod{p}$; $Y = g^y \pmod{p}$.
  - Given $(p, q, g, X, Y)$ find $g^{xy}$.

- **DDH problem:**
  - Pick $x, y, z \in \mathbb{R} \{1, \ldots, q\}$. Compute $X, Y$ as above
  - Given $(p, q, g, X, Y)$ distinguish $g^{xy}$ from $g^z$. 
How to solve the DLog problem

- In finite fields mod $p$:
  - Brute force (guess $x$) – $\Theta(q)$
  - Baby-step-giant-step: memory/computation tradeoff; $O(\sqrt{q})$
  - Pohlig-Hellman: small factors of $q$; $O(\log_p q (\log q + \sqrt{p}))$
  - Pollard-Rho (+PH): $O(\sqrt{p})$ for biggest factor $p$ of $q$
  - NFS, Pollard Lambda, ...
  - Index Calculus: $\exp\left((\ln q)^{\frac{1}{3}}(\ln(\ln(q)))^{\frac{2}{3}}\right)$

- Elliptic curves
  - Generic: best case is BSGS/Pollard-Rho
  - Some progress on Index-Calculus attacks recently
## Parameter Size vs. Security

### ANSSI

<table>
<thead>
<tr>
<th>Date</th>
<th>Sym.</th>
<th>RSA modulus</th>
<th>DLog Key</th>
<th>DLog Group</th>
<th>EC GF(p)</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;2020</td>
<td>100</td>
<td>2048</td>
<td>200</td>
<td>2048</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>&lt;2030</td>
<td>128</td>
<td>2048</td>
<td>200</td>
<td>2048</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>&gt;2030</td>
<td>128</td>
<td>3072</td>
<td>200</td>
<td>3072</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

### BSI

<table>
<thead>
<tr>
<th>Date</th>
<th>Sym.</th>
<th>RSA modulus</th>
<th>DLog Key</th>
<th>DLog Group</th>
<th>EC GF(p)</th>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015</td>
<td>128</td>
<td>2048</td>
<td>224</td>
<td>2048</td>
<td>224</td>
<td>SHA-224+</td>
</tr>
<tr>
<td>2016</td>
<td>128</td>
<td>2048</td>
<td>256</td>
<td>2048</td>
<td>256</td>
<td>SHA-256+</td>
</tr>
<tr>
<td>&lt;2021</td>
<td>128</td>
<td>3072</td>
<td>256</td>
<td>3072</td>
<td>256</td>
<td>SHA-256+</td>
</tr>
</tbody>
</table>
Using Assumptions

- Implicitly used for all the primitives you have ever heard of
- Take ElGamal encryption:
  - Setup: \( N \)-bit prime \( q \), \( L \)-bit prime \( p \) with \( q \mid (p - 1) \)
    Generator \( g \) such that \( \text{Order}(g \mod p) = q \)
    \[
g^q = kp + 1 \text{ for some } k \text{ and } g^m \neq np + 1 \text{ for any } n
    \]
  - Secret key: random \( sk \in \{1, \ldots, q - 1\} \)
  - Public key: \( pk = g^{sk} \mod p \)

DLog: you can’t compute \( sk \) from \( pk \)
**Using Assumptions (2)**

- Implicitly used for all the primitives you have ever heard of

- Take ElGamal encryption:
  - Setup: \( N \)-bit prime \( q \), \( L \)-bit prime \( p \) with \( q \mid (p - 1) \)
    Generator \( g \) such that \( \text{Order}(g \mod p) = q \)
  - Secret key: random \( sk \in \{1, \ldots, q - 1\} \)
  - Public key: \( pk = g^{sk} \pmod{p} \)
  - Encryption: pick random \( r \), output: \( (g^r, M \cdot pk^r) \pmod{p} \)
  - Decryption: \( \frac{M \cdot pk^r}{(g^r)^{sk}} = \frac{M \cdot (g^{sk})^r}{(g^r)^{sk}} \)

**CDH: can’t compute** \( g^{r \cdot sk} \) **from** \( g^r, g^{sk} \)
Using Assumptions (3)

- Implicitly used for all the primitives you have ever heard of
- Take Diffie-Helman key exchange (2-party):
  - Setup: $p, q, g$ as before
    - Alice
      - Pick $a$; $A = g^a$
      - Compute: $K = B^a$
    - Bob
      - Pick $b$; $B = g^b$
      - Compute: $K = A^b$

DDH: can’t distinguish $K$ from random, given $A, B$
PART IV
SECURITY MODELS
Ideal Provable Security

- Given protocol $\pi$, assumptions $H_1, \ldots, H_k$

"Real World" is hard to describe mathematically
**Provable Security**

- **Two-step process:**
  
  ![Diagram](image)

- **Real world using** $\pi$
- **Modelled world using** $\pi$
Provable Security

Real world using $\pi$

Proof under $H_1, \ldots, H_k$

Ideal world
COMPONENTS OF SECURITY MODELS

- Adversarial à-priori knowledge & computation:
  - Who is my adversary? (outsider, malicious party, etc.)
  - What does my adversary learn?

- Adversarial interactions (party-party, adversary-party, adversary-adversary – sometimes)
  - What can my adversary learn
  - How can my adversary attack?

- Adversarial goal (forge signature, find key, distinguish Alice from Bob)
  - What does my adversary want to achieve?
GAME-BASED SECURITY

- Participants
  - Adversary $\mathcal{A}$ plays a game against a challenger $\mathcal{C}$
  - Adversary = attacker(s), has all public information
  - Challenger = all honest parties, has public information and secret information

- Attack
  - Oracles: $\mathcal{A}$ makes oracle queries to $\mathcal{C}$ to learn information
  - Test: special query by $\mathcal{A}$ to $\mathcal{C}$ to which $\mathcal{A}$ responds sometimes followed by more oracle queries
  - Win/Lose: a bit output by $\mathcal{C}$ at the end of the game
**Canonical Game-Based Security**

- **Setup**: generate game parameters $s/p\text{Par}$
- **Learn**:
  - $A$ queries oracles; $C$ answers using $s$
  - $\text{ChGen}$: $C$ generates challenge $\text{chg}^*$
- **Result**: $C$ learns whether $A$ has won or lost

---

**Game Structure**

- Setup: generate game parameters $s/p\text{Par}$
- Learn: $A$ queries oracles; $C$ answers using $s$
- ChGen: $C$ generates challenge $\text{chg}^*$
- Result: $C$ learns whether $A$ has won or lost
**Example 1: Signature Schemes**

- **Intuition:** A signature scheme \((KGen, \text{Sign}, \text{Vf})\) is secure if and only if:
  - \(A\) should not be able to forge signatures

- **Formal security definition:** UNF-CMA

\[
\begin{align*}
(sk, pk) &\leftarrow KGen(1^\alpha) \\
(m, \sigma) &\leftarrow A^{OSign(*)^N}(1^\alpha, pk) \\
\text{Set} : L = \{m_i, \sigma_i\}_{i=1,...,N} &\text{ with } \sigma_i \leftarrow OSign(m_i) \\
A \text{ wins iff: } \text{Vf}(pk, m, \sigma) = 1 \text{ and } \{m, *\}/ \in L
\end{align*}
\]

\[
\begin{align*}
\text{OSign(*)} \\
\text{On input } m, \text{ set } \sigma \leftarrow \text{Sign}(sk, m) \\
\text{Output } \sigma
\end{align*}
\]
**Example 2: PKE**

- Intuition: a PK encryption scheme (KGen, Enc, Dec) is secure if and only if:
  - $\mathcal{A}$ should not be able to learn encrypted messages

- Formally defining this (without decryptions):

\[
\begin{align*}
(sk, pk) & \leftarrow \text{KGen}(1^\alpha) \\
 m & \leftarrow_R M \\
 c & \leftarrow \text{Enc}(pk, m) \\
 m' & \leftarrow A(1^\alpha, pk, c) \\
 A \text{ wins iff: } m = m'
\end{align*}
\]
**Example 2: PKE**

- **Intuition:** a PK encryption scheme \((KGen, Enc, Dec)\) is secure if and only if:
  - \(A\) should not be able to learn encrypted messages

- **What if** \(A\) can learn some ciphertext/plaintext tuples?

\[
\begin{align*}
(s_k, p_k) &\leftarrow KGen(1^\alpha) \\
\text{ready} &\leftarrow A^{ODec(\ast)^N}(1^\alpha, p_k) \\
m &\leftarrow_R M \\
c &\leftarrow Enc(p_k, m) \\
m' &\leftarrow A^{ODec'i(\ast)^M}(1^\alpha, p_k, c) \\
A \text{ wins iff: } m &= m'
\end{align*}
\]

**ODec(\ast)**
- On input \(c'\), output
  - \(m \leftarrow \text{Dec}(s_k, c')\)

**ODec'(\ast)**
- On input \(c' \neq c\), output
  - \(m \leftarrow \text{Dec}(s_k, c')\)
- Else, output \(\bot\)**
Example 2: PKE

- Intuition: a PK encryption scheme (KGen, Enc, Dec) is secure if and only if:
  - $\mathcal{A}$ should not be able to learn encrypted messages
  
  What if $\mathcal{A}$ can learn a single bit of the message?

  1 bit can make a difference in a small message space!

  - $\mathcal{A}$ should not be able to learn even 1 bit of an encrypted message
EXAMPLE 2: PKE

- Intuition: a PK encryption scheme \((\text{KGen, Enc, Dec})\) is secure if and only if:
  - \(\mathcal{A}\) must not learn even 1 bit of an encrypted message

- Formal definition: IND-CCA

\[
\begin{align*}
(sk, pk) &\leftarrow \text{KGen}(1^\alpha) \\
(m_1, m_2) &\leftarrow A^{O\text{Dec}(\ast)^N}(1^\alpha, pk) \\
b &\leftarrow_R \{0,1\} \\
c &\leftarrow \text{Enc}(pk, m_b) \\
d &\leftarrow A^{O\text{Dec}(\ast)^M}(1^\alpha, pk, c) \\
A \text{ wins iff: } b &= d
\end{align*}
\]
MEASURING ADVERSARIAL SUCCESS

- Winning a game; winning condition:
  - Depends on relation $R$ on $(\ast, <\text{game}>)$, with $<\text{game}> = \text{full game input (of honest parties and } \mathcal{A})$
  - Finally, $\mathcal{A}$ outputs $x$, wins if $(x, <\text{game}>) \in R$

- Success probability:
  - What is the probability that $\mathcal{A}$ “wins” the game?
  - What is the probability measured over? (e.g. randomness in $<\text{game}>$, sometimes probability space for keys, etc.)

- Advantage of Adversary:
  - How much better is $\mathcal{A}$ than a trivial adversary?
TRIVIAL ADVERSARIES

Example 1: Signature unforgeability
- $\mathcal{A}$ has to output a valid signature for message $m$
- Trivial attacks: (1) guess signature (probability $2^{-|\sigma|}$)
  (2) guess secret key (probability $2^{-|sk|}$)
  (3) re-use already-seen $\sigma$
- Goal: $\mathcal{A}$ outputs valid signature for fresh message $m$

Example 2: Distinguish real from random
- $\mathcal{A}$ has to output a single bit: real (0) or random (1)
- Trivial attacks: (1) guess the bit (probability $1/2$)
  (2) guess secret key (probability $2^{-|sk|}$)
Adversarial Advantage

- **Forgery type games:**
  - \( \mathcal{A} \) has to output a string of a “longer” size
  - Best trivial attacks: guess the string or guess the key
  - Advantage:
    \[
    \text{Adv}[A] = \text{Prob}[A \text{ wins the game}]
    \]

- **Distinguishability-type games:**
  - \( \mathcal{A} \) must distinguish between 2 things: left/right, real/random
  - Best trivial attacks: guess the bit (probability \( \frac{1}{2} \))
  - Advantage (different ways of writing it):
    \[
    \text{Adv}[A] = \text{Prob}[A \text{ wins the game}] - \frac{1}{2}
    \]
    \[
    \text{Adv}[A] = 2 \left| \text{Prob}[A \text{ wins the game}] - \frac{1}{2} \right|
    \]
DEFINING SECURITY

- Exact security definitions:
  - Input: number of significant queries of $\mathcal{A}$, execution time, advantage of $\mathcal{A}$
  - Example definition:

    A signature scheme $(\text{KGen, Sign, Vf})$ is $(N, t, \varepsilon)$-unforgeable under chosen message attacks (UNF-CMA) if for any adversary $\mathcal{A}$, running in time $t$, making at most $N$ queries to the Signing oracle, it holds that:

    $$\text{Adv}[A] := \text{Prob}[A \text{ wins the game}] \leq \varepsilon$$

- If a scheme is $(N, t, 1)$-UNF-CMA, then the scheme is insecure!
DEFINING SECURITY

- Asymptotic security:
  - Consider behaviour of $\varepsilon$ as a function of the size of the security parameter $1^\alpha$:

A signature scheme $(KGen, \text{Sign}, Vf)$ is $(N, t, \varepsilon)$-unforgeable under chosen message attacks (UNF-CMA) if for any adversary $\mathcal{A}$, running in time $t$, making at most $N$ queries to the Signing oracle, it holds that:

$$\text{Adv}[\mathcal{A}] = \text{Prob}[\mathcal{A} \text{ wins the game}] \leq \varepsilon$$

The signature is $(N, t)$-unforgeable under chosen message attacks if for any adversary $\mathcal{A}$ as above, it holds:

$$\text{Adv}[\mathcal{A}] \leq \text{negl}(1^\alpha)$$
**Simulation-Based Definitions**

- **Game-based definitions**
  - Well understood and studied
  - Can capture attacks up to “one bit of information”
  - What else do we need?

- **Zero-Knowledge: “nothing leaks about...”**
  - Real world: “real” parties, running protocol in the presence of a “local” adversary
  - Ideal world: “dummy” parties, simulator that formalizes the most leakage allowed from the protocol
  - “Global” adversary: distinguisher real/ideal world – if simulator is successful, then real world leaks as much as ideal world
SECURITY MODELS – CONCLUSIONS

Requirements:

- **Realistic** models: capture “reality” well, making proofs meaningful
- **Precise** definitions: allow quantification/classification of attacks, performance comparisons for schemes, generic protocol-construction statements
- **Exact** models: require subtlety and finesse in definitions, in order to formalize slight relaxations of standard definitions

Provable security is an art, balancing strong security requirements and security from minimal assumptions
Part V
Proofs of Security
GAME HOPPING

- Start from a given security game $G_0$
- Modify $G_0$ a bit (limiting $\mathcal{A}$) to get $G_1$
- Show that for protocol $\pi$, games $G_0$ and $G_1$ are equivalent (under assumption A), up to negligible factor $\varepsilon_1$:
  \[ G_0 \equiv_{\varepsilon_1} G_1 : \text{Prob}[A \text{ wins } G_0] \leq \text{Prob}[A \text{ wins } G_1] + \varepsilon_1 \]
- Hop through $G_2, G_3, \ldots, G_n$ (such that $G_{i-1} \equiv_{\varepsilon_i} G_i$ for all $i$)
- For last game $G_n$ find $\text{Prob}[A \text{ wins } G_n]$; then:
  \[ \text{Prob}[A \text{ wins } G_0] \leq \sum_{i=1}^{n} \varepsilon_i + \text{Prob}[A \text{ wins } G_n] \]
**Proving** $G_{i-1} \cong_{\varepsilon_i} G_i$

- **Method 1:** Reduce game indistinguishability to assumption or hard problem
  - If there exists a distinguisher $\mathcal{A}$ between $G_{i-1}$ and $G_i$ winning with probability $\frac{1}{2} + \delta$ then there exists an adversary $\mathcal{B}$ against assumption $H_1$ winning with probability $\delta' = f(\delta)$
  - So, $\text{Prob}[A \text{ wins } G_0] - \text{Prob}[A \text{ wins } G_1] \leq \delta + \delta' =: \varepsilon_1$

- **Method 2:** Reduce “difference” between games to assumption or hard problem
  - By construction, $\mathcal{A}$ can win $G_0$ more easily than $G_1$ (since $\mathcal{A}$ is more limited in $G_1$)
  - If there exists an adversary $B$ that can “take advantage of” the extra ability it has in $G_0$ to win w.p. $\text{Prob}[A \text{ wins } G_1] + \delta$, then there exists $B$ against $H_1$ winning w.p. $\delta'...$ (as above)
GAME EQUIVALENCE & REDUCTIONS

• Reduction: algorithm $R$ taking adversary $A$ against a game, outputting adversary $B$ against another game/hard problem

  \[ R^A \rightarrow B \]

• Intuition: if there exists an adversary $A$ against game $G$, this same adversary can be used by $R$ to obtain $B$ against $G'$

• $A$ interacts with challenger $C$ in $G$, $B$ interacts with $C'$ in $G'$

• In order to fully use $A$, $B$ needs to simulate $C$:
  • $A$ queries $C$ in game $G$: $B$ must answer query
  • $A$ sends challenge input to $C$: $B$ must send challenge
  • $A$ answers challenge: $B$ uses response in game $G'$
PART VI
AN EXAMPLE
Secure Symmetric-key Authentication

- Alice wants to authenticate to Bob, with whom she shares a secret key

Alice

\[ K \]

Bob

\[ \text{seed} \]

Choose

\[ \text{chg} \leftarrow \text{PRG(seed)} \]

\[ \text{rch} \leftarrow \text{PRF}_K(\text{chg}) \]

Verify:

\[ \text{rch} = \text{PRF}_K(\text{chg}) \]
SECURITY OF AUTHENTICATION

Nobody but Alice must authenticate to Bob

- Who is my adversary?
  - A man-in-the-middle

- What can they do?
  - Intercept messages, send messages (to Alice or Bob), eavesdrop

- What is their goal of $\mathcal{A}$?
  - Make Bob accept $\mathcal{A}$ as being Alice
TRIVIAL ATTACKS: RELAY

- Relay attacks bypass any kind of cryptography: encryption, hashing, signatures, etc.

- Countermeasure: distance bounding (we’ll see it later)
**Secure Authentication: Definition**

- **Session ID:** tuple \(< \text{chg}, \text{rsp} >\) used between partners

- **Oracles:**
  - \text{NewSession}(\ast): input either \(P_1 = \text{Alice}\) or \(P_2 = \text{Bob}\)
    
    outputs session “handle” \(\pi\)

  - \text{Send}(\ast,\ast): input handle \(\pi\) and message \(m \in M \cup \{\text{Prompt}\}\)
    
    transmits \(m\) to partner in \(\pi\), outputs \(m'\)

  - \text{Result}(\ast): input a handle \(\pi\) with partner \(P_2\)
    
    outputs 1 if \(P_2\) accepted authentication in \(\pi\),
    
    0 if \(P_2\) rejected, and ∅ otherwise
Secure Authentication: Game

**Game ImpSec:**

\[ k \leftarrow_R \text{KSpace}(1^\alpha) \]
\[ \text{seed} \leftarrow_R \text{SSpace}(1^\alpha) \]
\[ \text{done} \leftarrow A^{\text{NewSession(\#),Send(\#,\#),Result(\#)}(1^\alpha)} \]

\( \mathcal{A} \) wins iff \( \exists \pi \) output by \( \text{NewSession}(P_2) \) such that:

- \( \text{Result}(\pi) = 1 \);
- There exists no \( \pi' \) output by \( \text{NewSession}(P_1) \) such that \( \text{sid}(\pi) = \text{sid}(\pi') \)


- Protocol is \( (N_1, N_2, \varepsilon) \)-impersonation secure iff. no adversary \( \mathcal{A} \) using \( N_i \) sessions with \( P_i \) wins w.p. \( \geq \varepsilon \).

\[ \text{Adv}[A] := \text{Prob}[A \text{ wins}] \]
PRGs and PRFs

**Choose**

\[ \text{chg} \leftarrow \text{PRG}(\text{seed}) \]

**Verify:**

\[ \text{rsp} = \text{PRF}_K(\text{chg}) \]

- **Pseudorandomness of PRG:**
  
  \[ \text{key} \leftarrow_R \text{Kspace} \]
  
  \[ d \leftarrow A^{\text{Eval}_b()} \]
  
  \[ A \text{ wins iff. } d = b \]

\[ \text{Eval}_b(): \]

- if \( b = 0 \), return \( \text{Rand}() \)
- else, return \( \text{PRG}(\text{key}) \)
PRGs and PRFs

\[ K \quad \text{Alic} \quad e \quad \text{Bob} \quad \text{seed} \quad K \]

Choose
\[ \text{chg} \leftarrow \text{PRG}(\text{seed}) \]

\[ \text{rsp} \leftarrow \text{PRF}_K(\text{chg}) \]

Verify:
\[ \text{rsp} = \text{PRF}_K(\text{chg}) \]

- Pseudorandomness of PRF:
  \[ \text{key} \leftarrow_R \text{Kspace} \]
  \[ d \leftarrow A^{\text{Eval}_b()} \]
  \[ A \text{ wins iff. } d = b \]

\[ \text{Eval}_b(): \]
  \[ \text{choose } x \leftarrow_R X \]
  \[ \text{if } b = 0, \text{ return Rand}(x) \]
  \[ \text{else, return PRF}_{\text{key}}(x) \]
PROVING SECURITY

Choose
chg ← PRG(seed)

rsp ← PRFₖ(chg)

Verify:
rsp = PRFₖ(chg)

Intuition:
- If the PRG is good, then each chg is (almost) unique (up to collisions)
- If the PRF is good, then each rsp looks random to adversary
- Unless adversary relays, no chance to get right answer
Proof, step 1:
- Game $G_0$: Game ImpSec
- Game $G_1$: Replace $chg$ output by $P_2$ by random
- Equivalence: $G_0 \cong G_1$: if there exists $\varepsilon$-distinguisher $\mathcal{A}$ between $G_0$ and $G_1$, then there exists $\mathcal{B}$ against PRG winning w.p. $\varepsilon$
  - Basically the intuition is that if $\mathcal{A}$ can distinguish between the two games, he can distinguish real (PRG) from truly random challenges
Proving Security

Proof, equivalence $G_0 \cong G_1$:
- $\exists \varepsilon$-distinguisher $\mathcal{A}$ for $G_0 / G_1 \Rightarrow \exists \mathcal{B}$ winning PRG w.p. $\varepsilon$
  - Simulation: $\mathcal{B}$ chooses key $K \leftarrow R$ KSpace and simulate any requests to $\text{Send}(\pi, \text{Prompt})$ by $\text{Eval}_b()$ queries in PRG game
  - Finally $\mathcal{A}$ guesses either game $G_0$ ($\mathcal{B}$ outputs 1) or $G_1$ ($\mathcal{B}$ outputs 0)

Game PRG

<table>
<thead>
<tr>
<th>eval_{b}()</th>
<th>Seed \leftarrow R \text{ KSpace}</th>
<th>d \leftarrow B^{\text{eval}_{b}()}</th>
<th>B \text{ wins iff. } d = b</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval_{b}()</td>
<td></td>
<td>if $b = 0$, return Rand()</td>
<td>else, return PRG(key)</td>
</tr>
</tbody>
</table>
Proof, step 2:

- Game $G_0$: Game ImpSec
- Game $G_1$: Replace chg output by $P_2$ by random
- Game $G_2$: Abort if collision in chg

Equivalence: $G_1 \equiv G_2$: collisions in random strings occur in 2 different sessions w.p. $(1/2)^{|\text{chg}|}$. But we have a total of $N_2$ sessions, so the total probability of a collision is:

$$\binom{N_2}{2} 2^{-|\text{chg}|}$$
Proof, step 3:
- Game $G_0$: Game ImpSec
- Game $G_1$: Replace chg output by $P_2$ by random
- Game $G_2$: Abort if collision in chg
- Game $G_3$: replace honest responses by consistent, truly random strings
- Equivalence: $G_2 \cong G_3$: Similar to reduction to PRG, only this time it is to the pseudorandomness of the PRF.
PROVING SECURITY

Proof, step 4:

- Game $G_0$: Game ImpSec
- Game $G_1$: Replace chg output by $P_2$ by random
- Game $G_2$: Abort if collision in chg
- Game $G_3$: replace honest responses by consistent, truly random strings

At this point, the best the adversary can do is to guess a correct chg/rsp, i.e. $\text{Prob}[A \text{ wins } G_3] = N_1 \cdot 2^{-|\text{chg}|} + N_2 \cdot 2^{-|\text{rsp}|}$
PUTTING IT TOGETHER

\[ \Pr[A \text{ wins ImpSec}] \leq \Pr[A \text{ wins } G_1] + \text{Adv}[B \text{ against PRG}] \]

\[ \Pr[A \text{ wins } G_1] \leq \Pr[A \text{ wins } G_2] + \binom{N_2}{2} 2^{-|\text{chg}|} \]

\[ \Pr[A \text{ wins } G_2] \leq \Pr[A \text{ wins } G_3] + \text{Adv}[B \text{ against PRF}] \]

\[ \Pr[A \text{ wins } G_3] = N_1 \cdot 2^{-|\text{chg}|} + N_2 \cdot 2^{-|\text{rsp}|} \]
For every \((N_1, N_2, \varepsilon)\)-impersonation security adversary \(\mathcal{A}\) against the protocol, there exist:

- An \(\varepsilon_{\text{PRG}}\)-distinguisher against PRG
- An \(\varepsilon_{\text{PRF}}\)-distinguisher against PRF

such that:

\[
\varepsilon \leq \varepsilon_{\text{PRG}} + \varepsilon_{\text{PRF}} + \binom{N_2}{2} \ 2^{-|\text{chg}|} + N_1 \cdot 2^{-|\text{chg}|} + N_2 \cdot 2^{-|\text{rsp}|}
\]
**Provable Security**

- Powerful tool
- We can prove that a protocol is secure by design
- Captures generic attacks within a security model
- Can compare different schemes of same “type”
- 3 types of schemes:
  - Provably Secure
  - Attackable (found an attack)
  - We don’t know (unprovable, but not attackable)