The goal of secure program obfuscation is to make a program “unintelligible” while preserving its functionality. For decades, program obfuscation for general programs has remained an art, with all public general-purpose obfuscation methods known to be broken.

In this talk, we will describe new developments that for the first time provide a mathematical approach to the problem of general-purpose program obfuscation, where extracting secrets from the obfuscated program requires solving mathematical problems that currently take hundreds of years to solve on the world’s fastest computing systems. We will also discuss the implications of these developments.
Last Thursday’s lecture:
• Public key encryption
• IND-CPA / IND-CCA for public key schemes
• Trapdoor functions and permutations

Today’s lecture:
• Public key encryption from TDPs
• The RSA cryptosystem
Assignment 5 is TODAY!
Assignment 6 has been posted!
Recall: Trapdoor permutations (TDPs)

- Intuitively, a **trapdoor permutation family** is an **OWP family** with the additional property that each OWP in the family is **easy to invert** when given some special information called the **trapdoor**.
**Trapdoor permutation (TDP) families**

**Defn:** A trapdoor permutation (TDP) family is an infinite set \( \{\pi_k\}_{k\in K_e} \) in which each \( \pi_k : D_k \rightarrow D_k \) is a permutation, equipped with a quadruple of PPT algorithms \((\text{Gen}, \text{Samp}, \text{Eval}, \text{Inv})\) such that

- Easy to sample from family: \( \text{Gen} : 1^N \rightarrow K_e \times K_d \) is a randomized algorithm that, on input \( 1^s \), outputs \( (k, t_k) \) where \( t_k \) is the trapdoor for \( \pi_k \).

- Easy to sample from domains: \( \text{Samp} : K_e \rightarrow \bigcup D_k \) is a randomized algorithm that, on input \( k \in K_e \), outputs \( x \in D_k \).

- Easy to evaluate: \( \text{Eval} : K_e \times \bigcup D_k \) is a deterministic algorithm that, on input \( k \in K_e \) and \( x \in D_k \), outputs \( y = \pi_k(x) \).

- Easy to invert with trapdoor: \( \text{Inv} : K_e \times K_d \times \bigcup D_k \) is a deterministic algorithm that, on input any \( (k, t_k) \leftarrow \text{Gen}(1^s) \) and \( y \in D_k \), outputs \( x \in D_k \) such that \( y = \pi_k(x) \).

- Hard to invert without trapdoor: For every PPT attacker \( A \), there exists a negligible function \( \varepsilon : \mathbb{N} \rightarrow \mathbb{R}^+ \) such that
  \[
  \Pr[x \leftarrow A(k, y) \mid (k, t_k) \leftarrow \text{Gen}(1^s) \wedge x \leftarrow \text{Samp}(k) \wedge y \leftarrow \text{Eval}(k, x)] \leq \varepsilon(s).
  \]
Public-key bit encryption from TDPs

- Let \( \{\pi_k\}_{k \in \mathcal{K}} \) be a trapdoor permutation family with algorithms (Gen, Samp, Eval, Inv) such that each \( \pi_k \) has a corresponding hard-core predicate \( h_k \).

Q: Does such TDP family even exist?
A: Probably!? Or, at least, hopefully! (Why should we think it does?)
- Goldreich-Levin implies that if any TDP family exists, then there exists a TDP family in which each TDP has a hardcore predicate.
Public-key bit encryption from TDPs

- Let \( \{\pi_k\}_{k \in K_e} \) be a trapdoor permutation family with algorithms \((\text{Gen}, \text{Samp}, \text{Eval}, \text{Inv})\) such that each \( \pi_k \) has a corresponding hard-core predicate \( h_k \).

Define \((\text{Gen}, \text{Enc}, \text{Dec})\) as follows:
- \textbf{Gen}: \( 1^{n} \rightarrow K_e \times K_d \) is just the Gen algorithm for the TDP family
- \textbf{Enc}: \( K_e \times \{0,1\} \rightarrow D_k \) takes \((k,m) \in K_e \times \{0,1\}\) as input, chooses \( r \leftarrow \text{Samp}(k) \) such that \( h_k(r) = m \), and outputs \( c := \text{Eval}(k,r) \) requires 2 samples on average.
- \textbf{Dec}: \( K_e \times K_d \times D_k \rightarrow \{0,1\} \) takes \((k, t_k, c)\) as input and outputs \( h_k(\text{Inv}(k, t_k, c)) \)
Public-key bit encryption from TDPs

- Let \( \{ \pi_k \}_{k \in K_e} \) be a trapdoor permutation family with algorithms (Gen, Samp, Eval, Inv) such that each \( \pi_k \) has a corresponding hard-core predicate \( h_k \).

**Thm:** The tuple (Gen, Enc, Dec) just described is an IND-CCA2 secure public key encryption scheme with message space \( M = \{0, 1\} \).
Encryption for longer messages

Q: How do we extend bit encryption to longer messages?
A: Encrypt message bit-by-bit using the bit encryption scheme
   - But this is outrageously inefficient...

Q: How do we extend bit encryption to longer messages efficiently?
A: Very carefully...

- A candidate approach (with $M = D_k$):
  - Gen: $\mathbb{1}^n \rightarrow K_e \times K_d$ is the Gen algorithm for the TDP family
  - Enc: $K_e \times D_k \rightarrow D_k$ maps $(k, m)$ to $c := \text{Eval}(k, m)$
  - Dec: $K_e \times K_d \times D_k \rightarrow D_k$ maps $(k, t_k, c)$ to $m := \text{Inv}(k, t_k, c)$
Hybrid encryption from TDPs

- Let \( \{\pi_k\}_{k \in \mathcal{K}_e} \) be a trapdoor permutation family with algorithms (\( \text{Gen}, \text{Samp}, \text{Eval}, \text{Inv} \));
- Let \( (\text{Gen}', \text{Enc}', \text{Dec}') \) be an IND-CCA2 secure symmetric-key encryption scheme; and
- Let \( H: \{0,1\}^* \rightarrow \{0,1\}^* \) be a cryptographic hash function (modeled as a random oracle).

Define \( (\text{Gen}, \text{Enc}, \text{Dec}) \) as follows:

- \( \text{Gen}: \mathbb{N} \rightarrow \mathcal{K}_e \times \mathcal{K}_d \) is just the \( \text{Gen} \) algorithm for the TDP family;
- \( \text{Enc}: \mathcal{K}_e \times \{0,1\}^* \rightarrow \mathcal{D}_k \times \{0,1\}^* \) takes \( (k,m) \in \mathcal{K}_e \times \{0,1\}^* \) as input, chooses \( r \leftarrow \text{Samp}(k) \) and outputs \( c := (y,C) \), where \( y := \text{Eval}(k,r) \) and \( C := \text{Enc}'_{H(r)}(m) \);
- \( \text{Dec}: \mathcal{K}_e \times \mathcal{K}_d \times (\mathcal{D}_k \times \{0,1\}^*) \rightarrow \{0,1\}^* \) takes \( (k, t_k, (y, C)) \) as input, computes \( r := \text{Inv}(k, t_k, y) \), and outputs \( m := \text{Dec}'_{H(r)}(C) \).
Hybrid encryption from TDPs

Define $(\text{Gen}, \text{Enc}, \text{Dec})$ as follows:

- $\text{Gen} : \{0, 1\}^* \rightarrow K_e \times K_d$ is just the $\text{Gen}$ algorithm for the TDP family;
- $\text{Enc} : K_e \times \{0, 1\}^* \rightarrow D_k \times \{0, 1\}^*$ takes $(k, m) \in K_e \times \{0, 1\}^*$ as input, chooses $r \leftarrow \text{Samp}(k)$ and outputs $c := (y, C)$, where $y := \text{Eval}(k, r)$ and $C := \text{Enc}'_{H(r)}(m)$;
- $\text{Dec} : K_e \times K_d \times (D_k \times \{0, 1\}^*) \rightarrow \{0, 1\}^*$ takes $(k, t_k, (y, C))$ as input, computes $r := \text{Inv}(k, t_k, y)$, and outputs $m := \text{Dec}'_{H(r)}(C)$.
Thm: The tuple \((\text{Gen}, \text{Enc}, \text{Dec})\) described below is an IND-CCA2 secure public key encryption scheme with message space \(M = \{0,1\}^*\) in the random oracle model.

Define \((\text{Gen}, \text{Enc}, \text{Dec})\) as follows:

- \(\text{Gen}\): \(\mathbb{N} \rightarrow K_e \times K_d\) is just the \(\text{Gen}\) algorithm for the TDP family;
- \(\text{Enc}\): \(K_e \times \{0,1\}^* \rightarrow D_k \times \{0,1\}^*\) takes \((k,m) \in K_e \times \{0,1\}^*\) as input, chooses \(r \leftarrow \text{Samp}(k)\) and outputs \(c := (y,C)\), where \(y := \text{Eval}(k, r)\) and \(C := \text{Enc}_{H(r)}(m)\);
- \(\text{Dec}\): \(K_e \times K_d \times (D_k \times \{0,1\}^*) \rightarrow \{0,1\}^*\) takes \((k, t_k, (y,C))\) as input, computes \(r := \text{Inv}(k, t_k, y)\), and outputs \(m := \text{Dec}_{H(r)}(C)\).
Recall: The *eth* root problem

**Defn:** The eth root problem (aka the RSA problem) is:

Given \((n, e, a)\) such that

1. \(n = pq\) for distinct \(s\)-bit primes \(p\) and \(q\),
2. \(a \in \mathbb{Z}_n^*\), and
3. \(\gcd(e, \phi(n)) = 1\),

compute \(a^{1/e} \mod n\).

One possible solution: compute \(d := e^{-1} \mod \phi(n)\) and output \(a^d \mod n\).
Recall: RSA instance generator

**Defn:** An RSA instance generating algorithm $G$ is a PPT algorithm that, on input a security parameter $1^s \in 1^\mathbb{N}$, outputs a pair of distinct $s$-bit primes $(p, q)$ and $e \in \mathbb{Z}_{\phi(pq)}^\ast$. We write $(p, q, e) \leftarrow G(1^s)$ to indicate that $s$-bit primes $(p, q)$ and $e \in \mathbb{Z}_{\phi(pq)}^\ast$ are sampled from the output of $G$.

- We define hardness assumption for the $e$th root problem with respect to a particular fixed instance generator.
Recall: The $\ell$th root (RSA) assumption

Definition: Let $G$ be a fixed RSA instance generating algorithm. The $\ell$th root (RSA) assumption holds with respect to $G$ if, for every PPT algorithm $A$, there exists a negligible function $\epsilon: \mathbb{N} \rightarrow \mathbb{R}^+$ such that $\text{Adv}^{\text{RSA}, G}(A) \leq \epsilon(s)$.

\[
\text{Challenger (C)} \quad \text{Attacker (A)} \\
1 \leftarrow G(1^s) \\
(p, q, e) \leftarrow G(1^s) \\
N = p \cdot q \\
a \in \mathbb{Z}_N^* \\
b = a^e \mod N \\
(a, 1^s) \leftarrow G(1^s) \\
(N, e, b) \\
Let E be the event that $a \equiv a' \mod N$ \\
Define A's advantage to be $\text{Adv}^{\text{RSA}, G}(A) : = \Pr[E]$
The Strong RSA assumption

Defn: Let $G$ be a fixed RSA instance generating algorithm. The strong RSA assumption holds with respect to $G$ if, for every PPT algorithm $A$, there exists a negligible function $\varepsilon : \mathbb{N} \rightarrow \mathbb{R}^+$ such that $\text{Adv}_{\text{StrongRSA}}^G(A) \leq \varepsilon(s)$.

Challenger (C)

$(p, q, e) \leftarrow G(1^s)$

(If $\gcd(e', \varphi(n)) \neq 1$, choose again)

$N := p \cdot q$

$a \in \mathbb{Z}_N$

$b := a^e \mod N$

Let $E$ be the event that $a \equiv a' \mod N$

Define $A$'s advantage to be $\text{Adv}_{\text{StrongRSA}}^G(A) := \Pr[E]$
The RSA trapdoor permutation family

- The RSA assumption is equivalent to the assumption that the $e$th root problem induces a trapdoor permutation family

$$K_e := \{(pq, e) \mid p, q \in \text{Primes}[s] \text{ with } p \neq q \text{ and } \gcd(\phi(pq), e) = 1\}; K_d := \mathbb{N}; \text{ and } D_k := \mathbb{Z}_{pq}^*$$

- $\text{Gen}(1^s)$ chooses distinct primes $p, q \in \text{Primes}[s]$ and $e \in \mathbb{Z}_{\phi(pq)}$. It outputs $k := (pq, e)$ and $t_k := e^{-1} \mod \phi(pq)$
- $\text{Samp}(k)$ outputs $x \in \mathbb{Z}_{pq}^*$
- $\text{Eval}(k, x)$ outputs $y := x^e \mod pq$
- $\text{Inv}(k, t_k, y)$ outputs $x := y^d \mod pq$
"Textbook" (a.k.a. naïve) RSA

- Many textbooks (and courses) describe RSA encryption as a direct application of the RSA trapdoor permutation to a "message" (usually a symmetric key)

Q: Is this a secure public-key encryption scheme?
A: NO! NO! NO! Don't ever do this! (Seriously, don't do it!)
(If you do this and I find out, I will retroactive fail you in this course!)
- Meet-in-the-middle attacks
- Short message-small exponent attacks
- CRT-based attacks
- Related message attacks
Meet-in-the-middle attack

- Suppose $k \in_r \{0, 1\}^{128}$ is a random 128-bit symmetric key

Q: Given $(N, e)$ and $c := k^e \mod N$, how hard is it to recover $k$?

A: It is sometimes very easy!

- If $k \equiv k_1 \cdot k_2 \mod N$ where $\lg k_1 \approx \lg k_2 \approx 2^{64}$, then $k_2^e \equiv c \cdot m_2^{-e} \mod N$

Idea: Precompute a lookup table $L := \{1^e, 2^e, 3^e, \ldots, (2^{64})^e\} \mod N$

For each $a \in L$, compute $b := c \cdot a^{-1} \mod N$ and check if $b \in L$; if so, then $k \equiv a \cdot b \mod N$!
Meet-in-the-middle attack

- Computing $L := \{1^e, 2^e, 3^e, \ldots, (2^{64})^e\}$ requires about $2^{64} = \sqrt{2^{128}}$ exponentiations to power $e$ modulo $N$.

- A brute-force attack would have required about $2^{128}/2 = 2^{127}$ exponentiations to power $e$ modulo $N$. 
### Meet-in-the-middle attack

- **Toy example:** $N = 282,943 = 523 \cdot 541$ and $e = 3$
- **Message** $k$ **is an 8-bit symmetric key**
- **Ciphertext** is $c = k^e \equiv 206,824 \pmod{N}$

<table>
<thead>
<tr>
<th>$k^3$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1^3$</td>
<td>1</td>
</tr>
<tr>
<td>$2^3$</td>
<td>8</td>
</tr>
<tr>
<td>$3^3$</td>
<td>27</td>
</tr>
<tr>
<td>$4^3$</td>
<td>64</td>
</tr>
<tr>
<td>$5^3$</td>
<td>125</td>
</tr>
<tr>
<td>$6^3$</td>
<td>216</td>
</tr>
<tr>
<td>$7^3$</td>
<td>343</td>
</tr>
<tr>
<td>$8^3$</td>
<td>512</td>
</tr>
<tr>
<td>$9^3$</td>
<td>729</td>
</tr>
<tr>
<td>$10^3$</td>
<td>1,000</td>
</tr>
<tr>
<td>$11^3$</td>
<td>1,331</td>
</tr>
<tr>
<td>$12^3$</td>
<td>1,728</td>
</tr>
<tr>
<td>$13^3$</td>
<td>2,197</td>
</tr>
<tr>
<td>$14^3$</td>
<td>2,744</td>
</tr>
<tr>
<td>$15^3$</td>
<td>3,375</td>
</tr>
<tr>
<td>$16^3$</td>
<td>4,096</td>
</tr>
</tbody>
</table>

$\Rightarrow k \equiv 14 \cdot 15 = 210 \pmod{N}$  (indeed, $210^3 \equiv 206,824 \pmod{N}$)
Short message-small exponent attack

- Alice wishes to encrypt a short message, say $m=100$, under Bob's public key $k=(N,e)$
- If $N > m^e$, then no modular reduction occurs!
- In this case, given $c := m^e \mod n$, it is easy to compute $m = c^{1/e}$ in $\mathbb{Z}$!
Recall: Chinese Remainder Theorem (CRT)

Chinese Remainder Theorem: Let $n_1, n_2, \ldots, n_k$ be positive integers with $\gcd(n_i, n_j) = 1$ whenever $i \neq j$, and let $N = n_1 \cdot n_2 \cdots n_k$.

Then the system of congruence relations

\[
x \equiv c_1 \mod n_1 \\
\vdots \\
x \equiv c_k \mod n_k
\]

has a unique solution in $\mathbb{Z}_N$.

For each $i = 1, \ldots, k$, set $y_i = (N/n_i)^{-1} \mod n_i$ and $z_i = (N/n_i) \cdot y_i \mod N$.

The unique solution is $x \equiv \sum z_i \cdot c_i \mod N$. 

$z_i \equiv 1 \mod n_i$ and $z_i \equiv 0 \mod n_j$ when $i \neq j$. 

Ryan Henry
CRT-based attack

- Suppose Bob, Carol, and David have distinct public keys \((N_B, 3), (N_C, 3),\) and \((N_D, 3)\)
- Alice wishes to encrypt the same message \(m\) for each of them
  - \(c_B := m^3 \mod N_B\)
  - \(c_C := m^3 \mod N_C\)
  - \(c_D := m^3 \mod N_D\)
- If \(\gcd(N_B, N_C) = \gcd(N_B, N_D) = \gcd(N_C, N_D) = 1\), then by CRT there exists unique \(c \in \mathbb{Z}_N\) such that \(c \equiv c_i \mod N_i\) for \(i \in \{B, C, D\}\)
- Solving for \(c\), we get \(m = c^{1/3}\) in \(\mathbb{Z}\) (Why?)
- If \(N_A, N_B, N_C\) aren't pairwise relatively prime, attacker still learns \(m\) (How?)
That's all for today, folks!