APPLICATIONS OF INDISTINGUISHABILITY OBfuscation

Mark Zhandry – Stanford University

*Joint work with Dan Boneh
Program Obfuscation

Intuition: Scramble a program
- Same functionality as original
- Hides all implementation details

Potential uses:
- IP protection
- Prevent tampering
- Cryptography
Applications

Crypto
- Give out program with embedded secrets
- Obfuscate to hide secrets
- Ex: symmetric key to public key encryption

Keygen:
- Publish $\text{Enc}_o$ as $\text{pk}$
  $\Rightarrow k$ remains secret
- Keep $k$ as $sk$
Virtual Black Box (VBB) Obfuscation [BGIRSYY’01]

What can we learn about $P$ from an obfuscation $P_o$?
- Output on any input
- Anything derivable from polynomial number of outputs

VBB Obfuscation: can’t learn anything else
Virtual Black Box (VBB) Obfuscation \([\text{BGIRSVY'01}]\)

What can we learn about \(P\) from an obfuscation \(P_o\)?

- Output on any input.
- Anything derivable from a polynomial number of outputs.

VBB Obfuscation: cannot learn anything else.

\[ b=0,1 \]

\[ b \]

Theorem ([BGI+’01]): Can’t achieve for all programs.
More on VBB Impossibility

BGI+ construct program $P$ with embedded secret $k$ where:

- **$k$** is secret even given black box access to $P$
- Given **any** program computing $P$, can recover $k$

Main takeaways:

- Need weaker notion of obfuscation
- Obfuscation alone cannot guarantee secret hiding

Example:

- Some encryption schemes cannot be obfuscated
- Perhaps specific encryption schemes can be obfuscated?
  - e.g. public key encryption schemes
Indist. Obfuscation (iO) [BGI⁺’01, GR’07]

If two programs have the same functionality, obfuscations are indistinguishable:

\[ P_1(x) = P_2(x) \quad \forall x \]

BGI⁺ counter example does not apply to iO

However, big questions: How to build? How to use?
Indistinguishability Obfuscation (iO)

Answer:
• [GGHRSW’13] First candidate iO construction
  • Functional encryption

Exploding field:
• [BR’13, BGKPS’13, …] Additional constructions
• [SW’13, GGHR’13, BZ’13, ABGSZ’13, …] Uses
  • Public key encryption, signatures, deniable encryption, multiparty key exchange, MPC, …
• [BCPR’13, MR’13, BCP’13, …] Further Investigation
Our Results

Non-interactive multiparty key exchange
  • First scheme without trusted setup

Efficient broadcast encryption
  • Constant size ciphertext and secret keys
  • First *distributed* system: users generate keys themselves

Efficient traitor tracing
  • Shortest secret keys, ciphertexts, known
  • Resolves open problem in Differential Privacy [DNR⁺09]
MULTIPARTY KEY EXCHANGE
(Non-Interactive) Multiparty Key Exchange

Public bulletin board

$K_{ABCD}$

$K_{ABCD}$

$K_{ABCD}$

$K_{ABCD}$
History

2 parties: Diffie Hellman Protocol [DH’76]

3 parties: Bilinear maps [Joux’2000]

n>3 parties: Multilinear maps [BS’03, GGH’13, CLT’13]
  • Requires trusted setup phase

Our work: n parties, no trusted setup
Prior Constructions for $n>3$

First achieved using multilinear maps [GGH’13,CLT’13]

• These constructions all require trusted setup before protocol is run
• Trusted authority can also learn group key
Starting point for our construction

Building blocks:
- One-way function $G: S \rightarrow X$
- Pseudorandom function (PRF) $F$

Shared key: $F_k(x_1, x_2, x_3, x_4)$ ← how to compute securely?
Introduce Trusted Authority (for now)

\[ P( x'_1, \ldots, x'_n, s, i ) \{ \]
\[ \quad \text{If } G(s) \neq x'_i, \text{ output } \bot \]
\[ \quad \text{Otherwise, output } F_k(x'_1, \ldots, x'_n) \]
\[ \} \]
First attempt

$$K_{ABCD} = P_{iO}(x_1, x_2, x_3, x_4, s_1, 1)$$

Problems:
- $k$ not guaranteed to be hidden using iO
- Still have trusted authority
Removing Trusted Setup

As described, our scheme needs trusted setup

Observation: Obfuscated program can be generated independently of publishing step

\[
P( x'_1, \ldots, x'_n, s, i ) \{
    \text{If } G(s) \neq x'_i, \text{ output } \bot \\
    \text{Otherwise, output } F_k(x'_1, \ldots, x'_n )
\}
\]

Untrusted setup: designate user 1 as “master party”
- generates \( P_{iO} \), sends with \( x_1 \)
Multiparty Key Exchange Without Trusted Setup

Security equivalent to security of previous scheme
Hiding $k$

Follow “punctured program” paradigm of SW’13

- Use pseudorandom generator for $G$
  
  $G: S \rightarrow X \quad |X| \gg |S|$
  
  $G(s), s \leftarrow S \text{ indist. from } x \leftarrow X$

- Use special “punctured PRF” for $F$ [BW’13, KPTZ’13, BGI’13, SW’13]

  Punctured key $k^z \Rightarrow \text{compute } F_k(x)$ everywhere but $z$

  $F((k, x))$ if $x \neq z$

  $\perp$ if $x = z$

Security: given $k^z$, cannot compute $t=F_k(z)$

Construction: GGM’84
Security of Our Construction

```
P( x'₁, ..., x'ₙ, s, i ) {
  If G(s) ≠ x'_i,
    output ⊥
  Otherwise,
    output F_k(x'₁, ..., x'ₙ )
}
```

Adversary’s goal:
Learn F_k(x₁, ..., xₙ)
Step 1: Replace $x_i$

**Real World**

\[
P( x'_1, ..., x'_n, s, i ) \{ \\
  \text{If } G(s) \neq x'_i, \\
  \quad \text{output } \perp \\
  \text{Otherwise,} \\
  \quad \text{output } F_k(x'_1, ..., x'_n) \\
\}
\]

**Alternate World 1**

\[
P( x'_1, ..., x'_n, s, i ) \{ \\
  \text{If } G(s) \neq x'_i, \\
  \quad \text{output } \perp \\
  \text{Otherwise,} \\
  \quad \text{output } F_k(x'_1, ..., x'_n) \\
\]

Security of $G \Rightarrow$ words indistinguishable
Step 1: Replace $x_i$

Observation:
Since $|X| >> |S|$, w.h.p. no $s, i$ s.t. $G(s) = x_i$

Never pass check when $x'_1, ..., x'_n = x_1, ..., x_n$
Step 2: Puncture

Alternate World 2

\[ k^z \]

\[
P( x'_1, ..., x'_n, s, i ) \{ \\
\quad \text{If } G(s) \neq x'_i, \\
\qquad \text{output } \bot \\
\quad \text{If } (x'_1, ..., x'_n) = z, \\
\qquad \text{output } \bot \\
\quad \text{Otherwise,} \\
\qquad \text{output } F_k(x'_1, ..., x'_n) \\
\}
\]

W.h.p. programs identical + iO ⇒ Worlds indistinguishable

Alternate World 1

\[ k \]

\[
P( x'_1, ..., x'_n, s, i ) \{ \\
\quad \text{If } G(s) \neq x'_i, \\
\qquad \text{output } \bot \\
\quad \text{Otherwise,} \\
\qquad \text{output } F_k(x'_1, ..., x'_n) \\
\} 
\]

Let \( z = (x'_1, ..., x'_n) \)
Security

Alternate World 2

\[ k^z P( x'_1, ..., x'_n, s, i ) \{
\text{If } G(s) \neq x'_i, \\
\quad \text{output } \bot \\
\text{If } (x'_1, ..., x'_n) = z, \\
\quad \text{output } \bot \\
\text{Otherwise,} \\
\quad \text{output } F_k(x'_1, ..., x'_n) \\
\} \]

Adversary’s goal: learn \( F_k(z) \)

Success in Real World \( \Rightarrow \) success in World 2

In World 2:
Adversary only sees \( k^z \)
\( \Rightarrow \) cannot learn \( F_k(z) \)

Let \( z = (x_1, ..., x_n) \)
Minimal Assumptions

Building blocks:

• iO

• Pseudorandom generator $G: S \rightarrow X$ ($|X| >> |S|$)

• Puncturable PRF $F: K \times X^n \rightarrow Y$

Our constructions can be built from iO and worst-case complexity assumptions
ACTIVE SECURITY
Active Notions of Security

Key exchange protocol may be used multiple times
- Adversary may take part as well (even multiple times)
Active Notions of Security

Implications for our scheme:

• Everyone must be ready to be “master party”
  ⇒ everyone must publish own program $P_i$

• Use lexicographically minimal program

Malicious programs may run on honest secrets!
Active Notions of Security

Potential attack:

- Step 1: Attacker creates and publishes malicious program:

\[
P( y_1, ..., y_n, s, i ) \{ \\
\text{output } s \\
\}
\]
Potential attack:

• Step 2: Attacker and Bob use attacker’s program:

\[ K_{BE}(Bob) = P_E(x_B, x_E, s_B, B) = s_B \]
Active Notions of Security

Potential attack:
• Step 3: Attacker steals Bob’s shared keys:

\[ K_{BE}(Bob) = s_B \]
Active Notions of Security

Potential attack:

• Step 4: Attacker can compute any future shared key:

\[ K_{AB} = P_A(x_A, x_B, s_B, B) \]
Problems with Basic Scheme

Malicious programs run on honest secrets

Ways to fix?

• Ensure programs are honest
  Problematic since program obfuscated
• Never run untrusted programs on secrets
  (Assume inputs to completely leak)
Our Solution

- Replace user secret with signing key for signature scheme
- Publish public key
- Input to program is signature on set of users

\[ F( pk_1, ..., pk_n, S, \sigma, i ) \{ \]
\[ \quad \text{If } \text{Ver}( pk_i, S, \sigma ) \text{ rejects, output } \perp \]
\[ \quad \text{Otherwise, output } F(k, pk_1, ..., pk_n) \]
\[ \} \]

Intuition: Even after seeing many signatures, cannot learn signature on challenge set

**Theorem**: iO + “constrained signature” + “constrained PRF” \[ \Rightarrow \] “semi-static” security

Build from iO  Intermediate sec. notion  [BW’13]: build from MLM
Or, build from iO
REDUCING PARAMETER SIZES
Reducing Parameter Sizes [ABGSZ'13]

Key exchange program:

\[
P(k; x'_1, ..., x'_n, s, i) \begin{cases} 
\text{If } G(s) \neq x'_i, \text{ output } \perp \\
\text{Otherwise, output } F(k, x'_1, ..., x'_n) 
\end{cases}
\]

Size of input: \(\Omega(n)\)

For circuits, size of program: \(\Omega(n)\)

- Also true for Turing Machines (less obvious)

To reduce program size, must reduce input size

\[\Rightarrow\text{Must derive key from small string}\]
Reducing Parameter Sizes

Idea: use hash of public values to derive key

\[ h \leftarrow H(x_1, ..., x_n) \]
\[ k \leftarrow F(k, h) \]

User supplies \( h \) to program

Question: How does user \( i \) prove \( h \) is correct?

- Need proof that \( h=H(x_1', ..., x_n') \) where \( x_i' = x_i \)
- Need proof to be small

Answer: Merkle Hash Trees
Merkle Hash Trees

Proof size: $O(\log n)$
Our Construction

\[
\begin{align*}
\text{k} \quad & P( h, \pi, x, s, i ) \{ \\
& \quad \text{If } \pi \text{ is an invalid proof for } (h,x,i), \text{ output } \bot \\
& \quad \text{If } G(s) \neq x, \text{ output } \bot \\
& \quad \text{Otherwise, output } F(k,h) \\
\}
\end{align*}
\]

Program size: \text{poly} (\log n)

Problem: false proofs exist (though hard to compute)

• Must use stronger notion of obfuscation: diO
Open Questions

Reduce program sizes using iO?

Other primitives from iO
- FHE?
- Multilinear maps?

Thanks!