MULTIPARTY KEY EXCHANGE FROM 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
  • Embed secrets into program
  • Obfuscate program, publish obfuscation
Virtual Black Box (VBB) Obfuscation [BGIRSVY’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 [BGIRSVY’01]

What can we learn about $P_0$ from an obfuscation $P_o$?
• Output on any input
• Anything derivable in polynomial number of outputs

VBB Obfuscation: can’t learn anything else

Theorem ([BGI+’01]): Can’t achieve for all programs

$b=0, 1$
Indist. Obfuscation (iO) [BGI⁺’01, GR’07]

If two programs have same functionality, obfuscations are indistinguishable

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

\[ P_{1,0} \approx P_{2,0} \]

BGI⁺ counter example does not apply to iO
Indistinguishability Obfuscation (iO)

An exploding field:

- **[GGH⁺’13]** First candidate iO construction
  - Built from multilinear maps
  - First application: functional encryption
- **[BR’13, BGK⁺’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 Strategy for Using iO

1. Start with scheme that works using VBB obfuscation

2. Tweak scheme, enhance other primitives
   - OWF $\Rightarrow$ PRG
   - PRF $\Rightarrow$ Punctured PRF
   - Sig $\Rightarrow$ Constrained Sig

3. Relax VBB to iO, and try to prove security
This talk: Non-interactive multiparty key exchange

Static security:
- First construction without trusted setup

Active notions security:
- Basic construction completely broken
- Second construction achieves semi-static security
STATIC SECURITY
Statically Secure 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 \([\text{GGH'13, CLT'13}]\)

- These constructions all require trusted setup \textbf{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: $F_k(x_1, x_2, x_3, x_4) \leftarrow$ how to compute securely?
Introduce Trusted Authority (for now)

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

\[ P_0(x_1, x_2, x_3, x_4, s_1, 1) \]

Problems:
- Can prove security using VBB obfuscation, not 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

\[
P_k(x'_1, ..., x'_n, s, i) \begin{cases} 
    \text{If } G(s) \neq x'_i, \text{ output } \bot \\
    \text{Otherwise, output } F_k(x'_1, ..., x'_n) 
\end{cases}
\]

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

Security equivalent to security of previous scheme
VBB $\rightarrow$ iO

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 $k^z \Rightarrow$ compute $F_k(z)$ everywhere but $z$
  $$k^z$$

  $x$ $\rightarrow$ $F$ $\rightarrow$ $F(k,x)$ 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

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

Adversary’s goal:
Learn \( F_k(x_1, ..., x_n) \)
Security of Our Construction

$$P( x'_1, \ldots, x'_n, s, i ) \{$$
  If $G(s) \neq x'_i$,
    output $\bot$
  Otherwise,
    output $F_k(x'_1, \ldots, x'_n)$
$$\}$$
Step 1: Replace $x_i$

Real World

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

Alternate World 1

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

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

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

Alternate World 1

$$P( x'_1, \ldots, x'_n, s, i ) \{$$
If $G(s) \neq x'_i$,
output $\perp$
Otherwise,
output $F_k(x'_1, \ldots, x'_n)$
$$\}$$

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

Alternate World 2

$$P(\ x'_1, \ldots, x'_n, \ s, \ i \ ) \{$$
- If $$G(s) \neq x'_i,$$
  - output $$\bot$$
- If $$(x'_1, \ldots, x'_n) = z,$$
  - output $$\bot$$
- Otherwise,
  - output $$F_k(x'_1, \ldots, x'_n)$$

Alternate World 1

$$P(\ x'_1, \ldots, x'_n, \ s, \ i \ ) \{$$
- If $$G(s) \neq x'_i,$$
  - output $$\bot$$
- Otherwise,
  - output $$F_k(x'_1, \ldots, x'_n)$$

Let $$z = (x_1, \ldots, x_n)$$

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

Alternate World 2

$P(\ x'_1, \ldots, \ x'_n, \ s, \ i \ ) \{ \ \begin{array}{l}
\text{If } G(s) \neq x'_i, \\
\quad \text{output } \bot \\
\text{If } (x'_1, \ldots, x'_n) = z, \\
\quad \text{output } \bot \\
\text{Otherwise,} \\
\quad \text{output } F_k(x'_1, \ldots, x'_n) \\
\end{array} \}$

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 $z = (x_1, \ldots, x_n)$
ACTIVE SECURITY
Active Notions of Security

Key exchange protocol may be used multiple times
- Adversary may take part as well (even multiple times)
- May also learn shared secrets of honest parties
Full (Adaptive) Security [FHKP’13]

RegHon(i): create new honest user with id $i$, give public value $x_i$ to adversary, keep secret $s_i$ hidden ($i$ never repeated)

RegCor(i, x): create new corrupt user $i$ with public value $x$

Extract(i): give $s_i$ to adversary, mark $i$ as corrupt

Reveal(S, i): have (honest) user $i$ compute shared key for set $S$

Chal(S): give adversary either shared or random key for (honest) set $S$

Adversary's goal: tell if challenge is real or random
Implications for Our Scheme

- Everyone must be ready to be “master party”
  \[ \Rightarrow \text{everyone must publish own program } P_i \]
- Use lexicographically minimal program

Malicious programs may run on honest secrets!
Attack on Our Scheme

Step 1: Attacker registers corrupt user with malicious program:

\[
P_1(x_1, \ldots, x_n, s, i) \{
\text{output } s
\}
\]
Attack on Our Scheme

Step 2: Attacker runs reveal query for himself and Bob

\[ K_{BE}(\text{Bob}) = P_3(x_2, x_3, s_2, 1) = s_2 \]
Attack on Our Scheme

Step 2: Attacker runs reveal query for himself and Bob

\[ K_{BE}(Bob) = s_2 \]
Attack on Our Scheme

Step 3: Attacker can compute any future shared key
- In particular, can distinguish shared key from random

\[ K_{AB} = P_3(x_1, x_2, s_2, 2) \]
Problems with Basic Scheme

Malicious programs run on honest secrets
Also applies to VBB obfuscation (not an iO problem)

Ways to fix?

• Ensure programs are honest
  Problematic since program obfuscated
• Never run untrusted programs on secrets
  (Assume inputs 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

\[ P(S, \{pk_i\}_{i \in S}, \sigma, i) \{
\begin{align*}
& \text{If } \text{Ver}( pk_i, S, \sigma ) \text{ rejects, output } \bot \\
& \text{Otherwise, output } F_k( S, \{pk_i\}_{i \in S} )
\end{align*}
\]  

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

**Theorem**: \( iO + \text{“constrained signature”} + \text{“constrained PRF”} \Rightarrow \text{“semi-static” security} \)

Build from iO or zaps  Intermediate sec. notion  [BW’13]: build from MLM  Or, build from iO
Constrained PRFs [BW’13, KPTZ’13, BGI’13]

Generalization of punctured PRFs to circuits $C$
Constrained key $k^c \Rightarrow$ compute $F_k(\cdot)$ anywhere $C$ evaluates to 1

Security:
Given $k^c$, cannot compute $t = F_k(x)$ for any $x$ where $C(x) = 0$

Construction:
- [BW’13] from multilinear maps (requires complexity leveraging)
- From iO (requires complexity leveraging)
Constrained Signatures

Constrained public key $pk^C$, no sigs on $x$ where $C(x)=0$

$$\text{Ver}(pk^C, x, \sigma) = \text{reject} \text{ for all } (x, \sigma) \text{ where } C(x)=0$$

Security: given signatures on messages where $C(x)=1$, cannot distinguish $pk$ from $pk^C$

Constructions:
- From iO (requires complexity leveraging)
- From zaps [GGHW’13] (no complexity leveraging)
Semi-static security

Adversary commits to honest set $T^*$

$\text{RegHon}(i)$: create new honest user with id $i$, give public value $x_i$ to adversary, keep secret $s_i$ hidden ($i$ never repeated)

$\text{RegCor}(i \notin T^*,x)$: create new corrupt user $i$ with public value $x$

$\text{Extract}(i)$: give $s_i$ to adversary, mark $i$ as corrupt

$\text{Reveal}(S \notin T^*,i)$: have (honest) user $i$ compute key for set $S$

$\text{Chal}(S \subseteq T^*)$: give adversary either shared or random key for (honest) set $S$
Security of Our Construction

\[ k_1 \]
\[ P(S, \{pk'_i\}_{i \in S}, \sigma, i) \{
    \text{If } \text{Ver}( pk'_i, S, \sigma ) \text{ rejects, output } \bot \\
    \text{Otherwise, output } F_{k_1}( S, \{pk'_i\}_{i \in S} )
\}

\text{Adversary's goal: Learn } F_{k_j}(S, \{pk_i\}_{i \in S}) \text{ for some } j, S \subseteq T^* 

Answer **Reveal** queries using \( sk_i \)
Step 1: Constrain Verification Keys

Let $c_{sig}(S) = 1$ iff $S \notin T^*$

Idea: constrain sig verification keys to $c_{sig}$
- No valid signature on sets $S \subseteq T^*$

All reveal queries only have $S \notin T^*$
- Only need signatures on $S$ where $S \notin T^*$
- Constrained verification key indistinguishable from real
Step 1: Constrain Verification Keys

Real World

\[ P(S, \{pk'_i\}_{i \in S}, \sigma, i) \{ \]

If \( \text{Ver}(pk'_i, S, \sigma) \) rejects,
output \( \bot \)

Otherwise,
output \( F_{k_1}(S, \{pk'_i\}_{i \in S}) \)

\[ \]

Alternate World 1

\[ P(S, \{pk'_i\}_{i \in S}, \sigma, i) \{ \]

If \( \text{Ver}(pk'_i, S, \sigma) \) rejects,
output \( \bot \)

Otherwise,
output \( F_{k_1}(S, \{pk'_i\}_{i \in S}) \)

Constrain sig security \( \Rightarrow \) words indistinguishable
Step 2: Constrain PRF Keys

Draw $\mathbf{pk}_{i_1} \ldots \mathbf{pk}_{i_H}$ first (as constrained verification keys)

Let $C_{\text{prf}}( S, \{\mathbf{pk}'_i\} ) = 1$ iff $S \not\subseteq T^*$ OR $\{\mathbf{pk}'_i\} \neq \{\mathbf{pk}_i\}_{i \notin S}$

Idea: constrain PRF keys to $C_{\text{prf}}$

- Claim: can still use in program

$$\begin{align*}
\mathbf{k} &\mapsto \mathcal{P}( S, \{\mathbf{pk}'_i\}_{i \in S}, \sigma, i ) \{ \\
&\quad \text{If Ver}( \mathbf{pk}'_i, S, \sigma ) \text{ rejects, output } \bot \\
&\quad \text{Otherwise, output } F_k( S, \{\mathbf{pk}'_i\}_{i \in S} ) \\
\} \\
C_{\text{prf}}( S, \{\mathbf{pk}'_i\} ) = 0 &\Rightarrow S \subseteq T^* \text{ AND } \{\mathbf{pk}'_i\} = \{\mathbf{pk}_i\}_{i \notin S} \\
&\Rightarrow \text{Ver}( \mathbf{pk}_i, S, \sigma ) \text{ always rejects}
\end{align*}$$
Step 2: Puncture

Alternate World 2

\[ P(S, \{pk'_i\}_{i \in S}, \sigma, i) \{ \]
If Ver( pk', S, \sigma ) rejects,
output \( \bot \)
If \( C_{\text{prf}}( S, \{pk'_i\}_{i \in S}) = 0 \),
output \( \bot \)
Otherwise,
output \( F_{k_1}( S, \{pk'_i\}_{i \in S}) \)

Alternate World 1

\[ P(S, \{pk'_i\}_{i \in S}, \sigma, i) \{ \]
If Ver( pk', S, \sigma ) rejects,
output \( \bot \)
Otherwise,
output \( F_{k_1}( S, \{pk'_i\}_{i \in S}) \)

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

Alternate World 2

Adversary’s goal:
Learn $F_{k_j}(S, \{pk_i\}_{i \in S})$ for some $j$, $S \subseteq T^*$

Success in RW $\Rightarrow$ success in W2

In World 2:
- Adversary only sees $k_j^{\mathsf{Cprf}}$
- $C_{\mathsf{prf}}(S, \{pk'_i\}_{i \in S}) = 0$
  $\Rightarrow$ cannot learn $F_{k_j}(S, \{pk_i\}_{i \in S})$
Open Questions

Reduce program sizes using iO?
  • Goal: \textbf{polylog}(N) (achievable with diO)

Semi-static security w/o complexity leveraging?

Adaptive security?

Other primitives from iO?
  • FHE
  • Multilinear maps

Thanks!