Annihilation Attacks for Multilinear Maps

Mark Zhandry - MIT

Joint work with Eric Miles, Amit Sahai
Multilinear Maps

Multilinear maps

Multilinear Maps

ABE

Multiparty NIKE

Broadcat Enc

Witness Encryption

GGH’13

CLT’13

GGH’15

FE

ORE

Obfuscation

Much, much more
Mmap Attacks

\[ \text{Multiparty NIKE} \rightarrow \text{Broadcast Enc} \rightarrow \text{FE} \rightarrow \text{OPF} \]

\[ \text{ABE} \rightarrow \text{Witness Encryption} \rightarrow \text{GGH'13} \rightarrow \text{CLT'13} \rightarrow \text{GGH'15} \]

\[ \text{Obfuscation} \rightarrow \text{Much, much more} \]

\[ \text{GGH’13a, CHLRS’15, BWZ’14, CGHLMR’15, HJ’15, BGHLST’15, Hal’15, CLR’15, MF’15, CLLT’15, CFLMR’16} \]
This Work

**Goal:** Understand if/why Obfuscation/Witness Enc/ORE actually resists attack
Background...
Obfuscation

Intuition: scramble a program
• Maintain functionality, hide implementation

“Industry accepted” security notion: iO

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

\[ P_{1,0} \approx P_{2,0} \]
High-level description GGH13

Level $i$ encoding of $x$: $\frac{x + gr}{z^i}$ “short”

- Add within levels: $\frac{x_1 + gr_1}{z^i} + \frac{x_2 + gr_2}{z^i} = \frac{(x_1 + x_2) + g(r_1 + r_2)}{z^i}$

- Multiply: $\frac{x_1 + gr_1}{z^i} \cdot \frac{x_2 + gr_2}{z^j} = \frac{(x_1x_2) + g(r_1x_2 + r_2x_1 + gr_1r_2)}{z^{i+j}}$

IsZero(level $k$ encoding $e$): test if $p_{z^+e}$ is “not too big”

- $p_{z^+} = \frac{hz^k}{g}$ “not too big”

- $p_{z^+} \frac{gr}{z^k} = hr$ “not too big”

- $p_{z^+} \frac{x + gr}{z^k} = \frac{hx}{g} + hr$ “big”
High-level description GGH13

Level i encoding of $x$: $\frac{x + g r}{z^i}$

$\text{IsZero}(\text{level } k \text{ encoding } e)$: test if $p_{z+e}$ is “not too big”

Intuition:
• Can eval arbitrary degree-$k$ polys on level-1 encodings, then zero test
• For any degree higher than $k$, zero test gives junk

For obfuscation, use “asymmetric” variant
• Enforces additional constraints on allowable polys
Background on Mmap Attacks

For current mmaps, when IsZero=True, also get algebraic element $hr$ that can be manipulated
- $r$ may contain info about plaintexts

Idea behind all known (classical) attacks:
- Generate several “related” zero encodings
- Manipulate top-level encodings to learn non-trivial information

All attacks respect level restrictions
Background on Mmap Attacks

Prior attacks:
- Generally require some “low-level” zero encodings
  \[\Rightarrow\text{multiply together to get “related” zero encodings}\]
- Extends to cases where no explicit low level zero encodings are given, but “effective” encodings of zero are given
- \[\exists\text{quantum attacks that don’t need low-level zeros [BS’15]}\]
Background on Mmap Attacks

Most applications need low level encodings
• Used for “rerandomization”
• Required by most applications
• Hence, these applications broken

Also required to use most assumptions
• Inc. those used to build iO (e.g. [GLSW’14])
• Proofs often broken as well, even if application isn’t
Background on Mmap Attacks

Attacking obfuscation/witness encryption/ORE?
• No explicit low level zero encodings needed for schemes
• Top level zero encodings may still be generated during use

What next? Either:
1. Completely break application?
2. Argue that application is secure?
   • All known reductions to “simple” assumptions require low-level zero encodings
   • Need alternate way to argue security
Restricted Black Box Fields

\[ F = \text{Field}, \ P = \text{class of polynomials on } n \text{ variables} \]

\[ p \in \mathcal{P} \]

\[ \text{IsZero}( \ p(a_1, a_2, \ldots, a_n) \ ) \]

\[ a_1, a_2, \ldots, a_n \in F \]

Generic Groups*:
\[ P = \{ \text{Linear functions} \} \]

Black Box Fields*:
\[ P = \{ \text{Polys with small algebraic circuits} \} \]

Symmetrix multilinear maps*:
\[ P = \{ \text{Degree } k \text{ polynomials} \} \]

Asymmetric multilinear maps*:
\[ P = \{ \text{More complicated restrictions} \} \]

* Often need greater functionality requirements for protocols. This model suffices for our discussion
Obfuscation in Restricted BBFs

( model used by [BR’14, BGKPS’14, AGIS’14, Z’15, AB’15, BMSZ’16] )

**Obfuscate(C):**

![Diagram of Obfuscate(C)]

**Eval(x):**

![Diagram of Eval(x)]

- If *IsZero* gives “True”, output 1
- If *IsZero* gives “False”, output 0

Unfortunately, restricted BBF does not capture mmap attacks
Refined Abstract Model for Mmap attacks

\[ a_1, a_2, ..., a_n \in \mathbb{F} \]
\[ r_1, r_2, ..., r_n \leftarrow \$ \mathbb{F} \]

Write \( p(a_1 + gr_1, ..., a_n + gr_n) = c + dg + ... \)

If \( c \neq 0 \), output “False”
If \( c = 0 \), output “True”, \( d \)

Efficient polys

\( q \in \mathbb{Q} \)

Refined Abstract Model

Unrestricted BBF

\( d_1 \, d_2 \, d_3 \, ... \)
Refined Abstract Model for Mmap attacks

• Seems to capture intuition behind attacks

Proof in refined model  \rightarrow  Heuristic evidence of security against current attacks

But keep in mind that:

Attack in refined model  \not\rightarrow  Attack on actual protocol

Not trivially
Prior work: obfuscation for evasive functions [MBSZ’16]

What if function being obfuscated is evasive?
• When running obfuscator on any point the adversary can come up with, \texttt{IsZero} always gives “False”

• [BMSZ’16]: The only way to get IsZero to be “True” is through honest executions

• For evasive functions, all attacks apparently blocked

• In particular, witness enc secure against known attacks

What about non-evasive settings?
Attacking Obfuscation [MSZ’16]

Thm: The branching program obfuscators in [BGKPS’14, PST’14, AGIS’14, BMSZ’16] do not satisfy iO in the refined abstract model

Also: translate abstract model attack into concrete attack when instantiated using GGH’13 mmaps
  • Small heuristic component
(Single input) Branching Programs

\[ x = 11001: \]
\[ p_x = A_{1,1} A_{2,0} A_{3,1} A_{4,1} A_{5,0} A_{6,0} A_{7,1} A_{8,1} \]

If \( p_x = 0 \), output 1, otherwise output 0
Thm ([BMSZ’16]): If level structure respected, only poly’s that evaluate to 0 correspond to honest evaluations*

* Assuming mild technical condition on BP
[BMSZ’16] In Abstract Model

Honest subset products

\[ p_x \in P \]

Efficient polys

\[ q \in Q \]

Unrestricted BBF

\[ d_1, d_2, d_3, \ldots \]

\[ S_{i,b} \leftarrow \$ F \]

Write \[ p_x(\{B_{i,b} + g S_{i,b}\}) = c + dg + \ldots \]

If \( c \neq 0 \), output “False”
If \( c = 0 \), \( d \)
Attack: Annihilating Polynomials

- All terms are rational functions in underlying randomness
  \( \Rightarrow \) each \( d \) is rational in underlying randomness
- Efficiency \( \Rightarrow \) only poly-many free variables
- Exponentially many inputs \( \Rightarrow \) exponentially many \( d \)
- Must be algebraic dependence among \( d \)
  \[ \exists \text{poly } q: q(d_1, d_2, \ldots) = 0 \]
- \( q \) will most likely depend on exact program obfuscated

Argument extends to any “purely algebraic” obfuscator
Attack: Annihilating Polynomials

$q$ are called “annihilating polynomials”

Goal: find annihilating polynomials for various programs

Problem: in general, annihilating polys hard to compute

**Thm ([Kay’09]):** Unless PH collapses, there are dependent rational functions for which the annihilating polynomial requires super-polynomial sized circuits

Question: Can annihilating polys be found for particular obfuscators/programs?
Step 1: Variable Renaming

\[
B_{i,b} + gS_{i,b} = \alpha_{i,b} R_i^{-1} A_{i,b} R_{i+1} + gS_{i,b}
\]

\[
= \alpha_{i,b} R_i^{-1} \left( A_{i,b} + gT_{i,b} \right) R_{i+1}
\]

\[
T_{i,b} = \alpha_{i,b}^{-1} R_i S_{i,b} R_{i+1}^{-1}
\]

For honest subset product polynomials, \(R_i\)'s will cancel out

\[
\Rightarrow p_x(B_{i,b} + gS_{i,b}) = p_x(A_{i,b} + gT_{i,b})
\]
Step 2: Look at $g^1$ Coefficient

Coefficient of $g^1$ in $p_x(A_{i,b} + gT_{i,b})$:

$$d_x = \begin{bmatrix}
\beta_x & T_{1,1} & A_{2,0} & A_{3,1} & A_{4,1} & A_{5,0} & A_{6,0} & A_{7,1} & A_{8,1} \\
\beta_x & A_{1,1} & T_{2,0} & A_{3,1} & A_{4,1} & A_{5,0} & A_{6,0} & A_{7,1} & A_{8,1} \\
\beta_x & A_{1,1} & A_{2,0} & T_{3,1} & A_{4,1} & A_{5,0} & A_{6,0} & A_{7,1} & A_{8,1} \\
\beta_x & A_{1,1} & A_{2,0} & A_{3,1} & A_{4,1} & A_{5,0} & A_{6,0} & A_{7,1} & T_{8,1}
\end{bmatrix} + \ldots$$

$$\beta_x = \begin{bmatrix}
\alpha_{1,1} & \alpha_{2,0} & \alpha_{3,1} & \alpha_{4,1} & \alpha_{5,0} & \alpha_{6,0} & \alpha_{7,1} & \alpha_{8,1}
\end{bmatrix}$$
Step 2: Look at $g^1$ Coefficient

Suppose “trivial” branching program: $A_{i,0}=A_{i,1}=A_i$

![Diagram of branching program]

$d_x = \begin{align*}
\beta_x & T_{1,1} \\
\beta_x & A_1 \\
\beta_x & A_1 \\
\beta_x & A_1 \\
\beta_x & A_1 & T_{3,1} \\
\beta_x & A_1 & T_{8,1} \\
\end{align*}

Only parts that depend on $x$

$U_{i,x_{\text{inp}(i)}}$
Step 3: More Variable Renaming

Suppose “trivial” branching program: \( A_{i,0} = A_{i,1} = A_i \)

\[ d_x = \beta_x (U_{1,x_2} + U_{2,x_4} + U_{3,x_2} + U_{4,x_1} + U_{5,x_3} + U_{6,x_4} + U_{7,x_2} + U_{8,x_5}) \]

Collect \( U \) that read same input bit:

\[ \gamma_x = V_{1,x_1} + V_{2,x_2} + V_{3,x_3} + V_{4,x_4} + V_{5,x_5} \]

Same treatment for \( \beta_x \):

\[ \beta_x = W_{1,x_1} W_{2,x_2} W_{3,x_3} W_{4,x_4} W_{5,x_5} \]
Step 4: Even More Variable Renaming

\[ \gamma_x = V_{1,x_1} + V_{2,x_2} + V_{3,x_3} + V_{4,x_4} + V_{5,x_5} \]

Linear algebra!

\[ e(i) = 0\ldots010\ldots0 \quad 0 = 0^n \]
\[ x \leq y: \quad x_i=1 \implies y_i=1 \]

\[ \gamma_x = \sum_{e(i) \leq x} \gamma_{e(i)} - (|x|-1) \gamma_0 \]

Now algebraic dependence is local

(E.g. can consider \( x \) that are non-zero at only first \( k \) bits)
Step 4: Even More Variable Renaming

\[ \gamma_x = \sum_{e(i) \leq x} \gamma_{e(i)} - |x| \gamma_0 \quad \quad \beta_x = \prod_{e(i) \leq x} \gamma_{e(i)} / \beta_0^{|x|} \]

\[ d_x = \beta_x \gamma_x \]

Consider \( x \) that are non-zero only in first \( k \) bits

- 2\(^k\) different \( d_x \)
- \( 2(k+1) \) degrees of freedom
- For \( k \geq 3 \), must be algebraic dependence
Step 5: Brute Force Search

Brute force search for annihilating poly

\[ q = (d_{000}d_{111})^2 + (d_{001}d_{110})^2 + (d_{010}d_{101})^2 + (d_{100}d_{011})^2 \]
\[ - 2d_{000}d_{111}d_{001}d_{110} - 2d_{000}d_{111}d_{010}d_{101} - 2d_{000}d_{111}d_{100}d_{011} \]
\[ - 2d_{001}d_{110}d_{010}d_{101} - 2d_{001}d_{110}d_{100}d_{011} - 2d_{010}d_{101}d_{100}d_{011} \]
\[ + 4d_{000}d_{011}d_{101}d_{111} + 4d_{111}d_{001}d_{010}d_{100} \]

Annihilation very particular to “trivial” program
• \( q \) will not annihilate on “most” programs
The Abstract Attack

• Branching programs:
  • “Trivial” program that always outputs $1$
  • Non-trivial program that always outputs $1$

\[
p_{000}, \ p_{001}, \ p_{010}, \ \ldots
\]
\[
\text{True, True, True, } \ldots
\]
\[
p_{x}( \{ B_{i,b} + g \ S_{i,b} \} )
= d_{x}g + \ldots
\]

Unrestricted BBF

\[
d_{000}, \ d_{001}, \ d_{010}, \ \ldots
\]
Extending to GGH’13 Candidate

Unfortunately, cannot directly test if \( q=0 \) in GGH’13
• If \( q \) annihilates, obtain element in ideal \( <g> \)
• \( <g> \) hidden, so cannot immediately test membership

Our attack:
• Evaluate \( q \) on many sets of inputs \( S_j \)
• Set up non-trivial program so that \( q=0 \) for each \( S_j \)
  \( \Rightarrow \) Obtain many \( x_j \) in \( <g> \), regardless of program
• Heuristically assume \( x_j \) span \( <g> \)
• Evaluate \( q \) on “test” set \( S^* \)
  • \( q \) annihilates on \( S^* \) iff trivial program
• Test if result is in \( <g> \) using the \( x_i \)
Further Extensions

So far, only discussed single input BMSZ’16

For dual input:
• Using same ideas, can reduce search to finite-size
• Can brute force annihilating polynomial in constant time
• Hasn’t found it yet... but still gives poly-time attack

Other obfuscators:
• [BGKPS’14, PST’14, AGIS’14]: similar analysis
• Particular attack fails for [GGHRSW’13]

Also attack ORE [BLRSZZ’15] over GGH’13
Takeaways

Old attacks: intuition about mmap security wrong
• Old abstract mmap invalid in presence of low-level zeros

Our attacks: intuition for obfuscation security (no low-level zeros) also wrong
• Old abstract mmap invalid even without low-level zeros
• Need to revisit all constructions using mmaps
• Need new ways to argue security

Future Work
• Extend attacks to other mmaps/obfuscators
• Defenses
Obfuscation