Linicrypt: A Model for Practical Cryptography

Brent Carmer **Mike Rosulek**

Oregon State University

Charles River Crypto Day
December 9, 2016

A Model for "Practical" Cryptography

Block cipher modes

$\underline{\text{CBC.Enc}_k(m_1, \ldots, m_n):}$

$c_0 \leftarrow \{0,1\}^\lambda$
for $i = 1$ to n:
$\quad c_i := E_k(m_i \oplus c_{i-1})$
return $(c_0, \ldots, c_n)$

$\underline{\text{OCB.Enc}_k(m_1, \ldots, m_n):}$

$L := E_k(0^\lambda)$
$R := E_k(N \oplus L)$
for $i = 1$ to n:
$\quad c_i := E_k(m_i \oplus \gamma_i L \oplus R) \oplus \gamma_i L \oplus R$
$\quad \vdots$
return $(c_0, \ldots, c_n)$

A Model for "Practical" Cryptography

Block cipher modes

$\underline{\text{CBC.Enc}_k(m_1, \ldots, m_n):}$

$\quad c_0 \leftarrow \{0,1\}^\lambda$
$\quad \text{for } i = 1 \text{ to } n:$
$\quad\quad c_i := E_k(m_i \oplus c_{i-1})$
$\quad \text{return } (c_0, \ldots, c_n)$

$\underline{\text{OCB.Enc}_k(m_1, \ldots, m_n):}$

$\quad L := E_k(0^\lambda)$
$\quad R := E_k(N \oplus L)$
$\quad \text{for } i = 1 \text{ to } n:$
$\quad\quad c_i := E_k(m_i \oplus \gamma_i L \oplus R) \oplus \gamma_i L \oplus R$
$\quad\quad \vdots$
$\quad \text{return } (c_0, \ldots, c_n)$

- Sample a random block

A Model for "Practical" Cryptography

Block cipher modes

$\underline{\text{CBC.Enc}_k(m_1, \ldots, m_n):}$

$c_0 \leftarrow \{0,1\}^\lambda$
for $i = 1$ to n:
$c_i := E_k(m_i \oplus c_{i-1})$
return $(c_0, \ldots, c_n)$

$\underline{\text{OCB.Enc}_k(m_1, \ldots, m_n):}$

$L := E_k(0^\lambda)$
$R := E_k(N \oplus L)$
for $i = 1$ to n:
$c_i := E_k(m_i \oplus \gamma_i L \oplus R) \oplus \gamma_i L \oplus R$
$\vdots$
return $(c_0, \ldots, c_n)$

- Sample a random block
- XOR blocks

A Model for "Practical" Cryptography

Block cipher modes

$\underline{\text{CBC.Enc}_k(m_1, \ldots, m_n)\text{:}}$

$c_0 \leftarrow \{0,1\}^\lambda$
for $i = 1$ to n:
  $c_i := E_k(m_i \oplus c_{i-1})$
return $(c_0, \ldots, c_n)$

$\underline{\text{OCB.Enc}_k(m_1, \ldots, m_n)\text{:}}$

$L := E_k(0^\lambda)$
$R := E_k(N \oplus L)$
for $i = 1$ to n:
  $c_i := E_k(m_i \oplus \gamma_i L \oplus R) \oplus \gamma_i L \oplus R$
  ⋮
return $(c_0, \ldots, c_n)$

- Sample a random block
- XOR blocks
- Call block cipher

A Model for "Practical" Cryptography

Block cipher modes

CBC.Enc$_k(m_1, \ldots, m_n)$:

$c_0 \leftarrow \{0,1\}^\lambda$

for $i = 1$ to n:

$c_i := E_k(m_i \oplus c_{i-1})$

return $(c_0, \ldots, c_n)$

OCB.Enc$_k(m_1, \ldots, m_n)$:

$L := E_k(0^\lambda)$

$R := E_k(N \oplus L)$

for $i = 1$ to n:

$c_i := E_k(m_i \oplus \gamma_i L \oplus R) \oplus \gamma_i L \oplus R$

$\vdots$

return $(c_0, \ldots, c_n)$

- Sample a random block
- XOR blocks
- Call block cipher
- $GF(2^\lambda)$ linear operation

A Model for "Practical" Cryptography

Hash-based signatures

Lamport.KeyGen():

  for $i = 1$ to n and $b \in \{0,1\}$:

    $sk_i^b \leftarrow \{0,1\}^\lambda$

    $vk_i^b = H(sk_i^b)$

Lamport.Sign(sk, m):

  return $(sk_1^{m_1}, \ldots, sk_n^{m_n})$

Winternitz.KeyGen():

  $sk \leftarrow \{0,1\}^\lambda$

  $vk = \underbrace{H(H(\cdots H}_{n}(sk)\cdots))$

Winternitz.Sign(sk, m):

  return $\underbrace{H(H(\cdots H}_{m}(sk)\cdots))$

A Model for "Practical" Cryptography

Hash-based signatures

Lamport.KeyGen():
   for $i = 1$ to n and $b \in \{0,1\}$:
      $sk_i^b \leftarrow \{0,1\}^\lambda$
      $vk_i^b = H(sk_i^b)$

Lamport.Sign(sk, m):
   return $(sk_1^{m_1}, \dots, sk_n^{m_n})$

Winternitz.KeyGen():
   $sk \leftarrow \{0,1\}^\lambda$
   $vk = \underbrace{H(H(\cdots H}_{n}(sk)\cdots))$

Winternitz.Sign(sk, m):
   return $\underbrace{H(H(\cdots H}_{m}(sk)\cdots))$

- Sample a random block

A Model for "Practical" Cryptography

Hash-based signatures

Lamport.KeyGen():

for $i = 1$ to n and $b \in \{0,1\}$:

$sk_i^b \leftarrow \{0,1\}^\lambda$

$vk_i^b = H(sk_i^b)$

Lamport.Sign(sk, m):

return $(sk_1^{m_1}, \ldots, sk_n^{m_n})$

Winternitz.KeyGen():

$sk \leftarrow \{0,1\}^\lambda$

$vk = \underbrace{H(H(\cdots H}_{n}(sk)\cdots))$

Winternitz.Sign(sk, m):

return $\underbrace{H(H(\cdots H}_{m}(sk)\cdots))$

- Sample a random block
- Call one-way function

A Model for "Practical" Cryptography

Hash-based signatures

Lamport.KeyGen():

for $i = 1$ to n and $b \in \{0,1\}$:

$sk_i^b \leftarrow \{0,1\}^\lambda$

$vk_i^b = H(sk_i^b)$

Lamport.Sign(sk, m):

return $(sk_1^{m_1}, \ldots, sk_n^{m_n})$

Winternitz.KeyGen():

$sk \leftarrow \{0,1\}^\lambda$

$vk = \underbrace{H(H(\cdots H}_{n}(sk)\cdots))$

Winternitz.Sign(sk, m):

return $\underbrace{H(H(\cdots H}_{m}(sk)\cdots))$

- Sample a random block
- Call one-way function

Modern hash-based signatures (e.g., SPHINCS) built from these components.

A Model for "Practical" Cryptography

Garbled circuits

GarbledANDGate(A, B, C, Δ):

output:

$$\begin{aligned} G_1 &= H(A, B) \oplus C \\ G_2 &= H(A, B \oplus \Delta) \oplus C \\ G_3 &= H(A \oplus \Delta, B) \oplus C \\ G_4 &= H(A \oplus \Delta, B \oplus \Delta) \oplus C \oplus \Delta \end{aligned}$$

Eval(A^*, B^*):

return $H(A^*, B^*) \oplus G_i$

(significantly simplified)

A Model for “Practical” Cryptography

Garbled circuits

GarbledANDGate(A, B, C, Δ):

output:

$$
\begin{aligned}
G_1 &= H(A, B) \oplus C \\
G_2 &= H(A, B \oplus \Delta) \oplus C \\
G_3 &= H(A \oplus \Delta, B) \oplus C \\
G_4 &= H(A \oplus \Delta, B \oplus \Delta) \oplus C \oplus \Delta
\end{aligned}
$$

Eval(A^*, B^*):

return $H(A^*, B^*) \oplus G_i$

(significantly simplified)

- XOR blocks
- Call hash function function

High-level overview

Linicrypt = class of algorithms that can:

- uniformly **sample** finite field elements,
- **query** a random oracle (input/output = field elements),
- apply **linear** combinations to field elements.

Captures a broad range of practical cryptographic techniques.

Linicrypt Highlights

Efficient Decidability:

Can decide, in polynomial time, whether two Linicrypt programs have indistinguishable outputs (in RO model).

$$\boxed{P_1} \stackrel{?}{\cong} \boxed{P_2}$$

Linicrypt Highlights

Efficient Decidability:

Can decide, in polynomial time, whether two Linicrypt programs have indistinguishable outputs (in RO model).

$$\boxed{P_1} \stackrel{?}{\cong} \boxed{P_2}$$

Synthesis:

Can express security of a Linicrypt program as a **existential formula** $\Rightarrow$ use SAT solver to discover programs that satsisfy a security condition.

Linicrypt Highlights

Efficient Decidability:
Can decide, in polynomial time, whether two Linicrypt programs have indistinguishable outputs (in RO model).

$$\boxed{P_1} \overset{?}{\cong} \boxed{P_2}$$

Synthesis:
Can express security of a Linicrypt program as a **existential formula** $\Rightarrow$ use SAT solver to discover programs that satsisfy a security condition.

Fine-Grained:
Possibility of lower-bounds on **concrete constants** (e.g., possible with 3 field elements, impossible with 2). Strong lower bounds in RO model.

Outline

1. **Linicrypt model & technical tools**
2. Synthesizing Linicrypt programs
3. Applications to Garbled Circuits
4. Future work, open problems

Model

Allowed Linicrypt operations:

$v_1 := \mathsf{input}[1]$ // input a field element

$v_2 \leftarrow \mathbb{F}$ // sample the field

$v_3 := H(v_2)$ // query RO with field elem

$v_4 := v_1 + 2\,v_3$ // fixed linear combination

return v_2, v_4 // return any number of vars

Model

Allowed Linicrypt operations:

$v_1 := \mathsf{input}[1]$	// input a field element
$v_2 \leftarrow \mathbb{F}$	// sample the field
$v_3 := H(v_2)$	// query RO with field elem
$v_4 := v_1 + 2\,v_3$	// fixed linear combination
return v_2, v_4	// return any number of vars

Similar models:

- **Minicrypt** [Impagliazzo95]: no linearity restriction
- **Generic Group Model** [Shoup97]: only linear operations "in the exponent", but can use *secret* coefficients
- **Arithmetic Model** [ApplebaumAvronBrzuska15]: no random oracle, constructions *oblivious* to choice of field

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Tools:

- **Algebraic representation** - output distribution of Linicrypt programs as collection of matrices/vectors.
- **Normal form** - find a canonical representation.
- **Basis changes** - reorder the variables.

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Tools:

- **Algebraic representation** - output distribution of Linicrypt programs as collection of matrices/vectors.
- **Normal form** - find a canonical representation.
- **Basis changes** - reorder the variables.

Algebraic representation: base variables

P:

$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

Algebraic representation: base variables

P:

$v_1 \leftarrow \mathbb{F}$

$v_2 \leftarrow \mathbb{F}$

$v_3 := v_1 + v_2$

$v_4 := H(v_3)$

$v_5 := v_4 + v_1$

$v_6 := H(v_5)$

return (v_4, v_5)

Def: A **base** variable is the result of a *sampling* step or *call to H*.

Algebraic representation: base variables

P:

$v_1 \leftarrow \mathbb{F}$

$v_2 \leftarrow \mathbb{F}$

$v_3 := v_1 + v_2$

$v_4 := H(v_3)$

$v_5 := v_4 + v_1$

$v_6 := H(v_5)$

return (v_4, v_5)

Def: A **base** variable is the result of a *sampling* step or *call to H*.

Every variable in **P** is a **linear function** of the base variables.

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} & & & \\ \\ \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:

$v_1 \leftarrow \mathbb{F}$

$v_2 \leftarrow \mathbb{F}$

$v_3 := v_1 + v_2$

$v_4 := H(v_3)$

$v_5 := v_4 + v_1$

$v_6 := H(v_5)$

return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & 0 \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:

$v_1 \leftarrow \mathbb{F}$

$v_2 \leftarrow \mathbb{F}$

$v_3 := v_1 + v_2$

$v_4 := H(v_3)$

$v_5 := v_4 + v_1$

$v_6 := H(v_5)$

return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & & & \\ 0 & 0 & 1 & 0 \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since every var in **P** is a linear function of the base variables, write them as a **matrix**.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since **the output of P** is a linear function of the base variables, write it as a matrix.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

Algebraic representation: program as matrix

Since **the output of P** is a linear function of the base variables, write it as a matrix.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Algebraic representation: program as matrix

Since **the output of P** is a linear function of the base variables, write it as a matrix.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_4 \\ v_6 \end{bmatrix}$$

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$\mathcal{M}$ does not account for the way **variables can be correlated via H.**

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at <u>this</u> linear combination of base variables, receive <u>that</u> combination of base variables."*

<u>P:</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ [\qquad] \mapsto [\qquad], \quad [\qquad] \mapsto [\qquad] \Big\}$$

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at this linear combination of base variables, receive that combination of base variables."*

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ [\qquad] \mapsto [\qquad], \quad [\qquad] \mapsto [\qquad] \Big\}$$

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at <u>this</u> linear combination of base variables, receive <u>that</u> combination of base variables."*

$\underline{\text{P:}}$
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \atop \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathcal{C} = \Big\{ \underline{\overset{v_1\ v_2\ v_4\ v_6}{[1\ \ 1\ \ 0\ \ 0]} \mapsto \overset{v_1\ v_2\ v_4\ v_6}{[0\ \ 0\ \ 1\ \ 0]}}, \quad [\qquad\qquad] \mapsto [\qquad\qquad] \Big\}$$

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at <u>this</u> linear combination of base variables, receive <u>that</u> combination of base variables."*

<u>P:</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ 1 \ 0 \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ 0 \ 1 \ 0] \end{array}, \quad [\qquad] \mapsto [\qquad] \right\}$$

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at <u>this</u> linear combination of base variables, receive <u>that</u> combination of base variables."*

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ 1 \ 0 \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ 0 \ 1 \ 0] \end{array}, \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ 0 \ 1 \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ 0 \ 0 \ 1] \end{array} \right\}$$

Algebraic representation: oracle constraints

An **oracle constraint** captures: *"query H at <u>this</u> linear combination of base variables, receive <u>that</u> combination of base variables."*

<u>P:</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 1\ 0\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 1\ 0] \end{array}, \quad \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 0\ 1\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 0\ 1] \end{array} \Big\}$$

Lemma: $\mathcal{M}$ & $\mathcal{C}$ completely characterize P!

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Tools:

- **Algebraic representation** - output distribution of Linicrypt programs as matrices.
- **Normal form** - remove "extra" oracle queries to generate a canonical representation.
- **Basis changes** - reorder the variables.

Normalize: unreachable oracle queries

What **linear constraints of base variables** does Adv know?

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 1\ 0\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 1\ 0] \end{array}, \quad \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 0\ 1\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 0\ 1] \end{array} \Big\}$$

Normalize: unreachable oracle queries

What **linear constraints of base variables** does Adv know?

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ \ 1 \ \ 0 \ \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ \ 0 \ \ 1 \ \ 0] \end{array}, \quad \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ \ 0 \ \ 1 \ \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ \ 0 \ \ 0 \ \ 1] \end{array} \right\}$$

- Every row of $\mathcal{M}$ is a known constraint on base vars

Normalize: unreachable oracle queries

What **linear constraints of base variables** does Adv know?

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \\ \end{array} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathcal{C} = \left\{ \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \\ \end{array} [1\ 1\ 0\ 0] \mapsto \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \\ \end{array} [0\ 0\ 1\ 0], \quad \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \\ \end{array} [1\ 0\ 1\ 0] \mapsto \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \\ \end{array} [0\ 0\ 0\ 1] \right\}$$

- Every row of $\mathcal{M}$ is a known constraint on base vars
- If oracle constraint has LHS $\in$ span(known),

Normalize: unreachable oracle queries

What **linear constraints of base variables** does Adv know?

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} v_1\ v_2\ v_4\ v_6 \\ [1\ 1\ 0\ 0] \end{array} \mapsto \begin{array}{c} v_1\ v_2\ v_4\ v_6 \\ [0\ 0\ 1\ 0] \end{array}, \begin{array}{c} v_1\ v_2\ v_4\ v_6 \\ [1\ 0\ 1\ 0] \end{array} \mapsto \begin{array}{c} v_1\ v_2\ v_4\ v_6 \\ [0\ 0\ 0\ 1] \end{array} \right\}$$

- Every row of $\mathcal{M}$ is a known constraint on base vars
- If oracle constraint has LHS $\in$ span(known), then add RHS

Normalize: unreachable oracle queries

What **linear constraints of base variables** does Adv know?

<u>P:</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 1\ 0\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 1\ 0] \end{array}, \quad \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1\ 0\ 1\ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0\ 0\ 0\ 1] \end{array} \right\}$$

- Every row of $\mathcal{M}$ is a known constraint on base vars
- If oracle constraint has LHS $\in$ span(known), then add RHS

In this example, first oracle constraint is **unreachable**.

Normalize: unreachable oracle queries

Claim: Pr[Adv calls H on unreachable query] $=$ negl.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ 1 \ 0 \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ 0 \ 1 \ 0] \end{array}, \quad \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [1 \ 0 \ 1 \ 0] \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ [0 \ 0 \ 0 \ 1] \end{array} \right\}$$

Normalize: unreachable oracle queries

Claim: Pr[Adv calls H on unreachable query] $=$ negl.

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 := H(v_3)$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \end{array}, \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

- Query 1 is **unreachable**

Normalize: unreachable oracle queries

Claim: Pr[Adv calls H on unreachable query] $=$ negl.

P:

$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 \leftarrow \mathbb{F}$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_4 & v_6 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \cancel{\begin{array}{c} \begin{matrix} v_1 & v_2 & v_4 & v_6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{matrix} v_1 & v_2 & v_4 & v_6 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \end{array}}, \begin{array}{c} \begin{matrix} v_1 & v_2 & v_4 & v_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{matrix} v_1 & v_2 & v_4 & v_6 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

- Query 1 is **unreachable**
- Remove that oracle constraint $\Rightarrow$ negligible effect on distinguisher

Normalize: useless oracle queries

Other kinds of oracle constraints can be removed, too!

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 \leftarrow \mathbb{F}$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

Normalize: useless oracle queries

Other kinds of oracle constraints can be removed, too!

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 \leftarrow \mathbb{F}$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

- v_6 never even used in the program, clearly can be removed

Normalize: useless oracle queries

Other kinds of oracle constraints can be removed, too!

P:
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 \leftarrow \mathbb{F}$
$v_5 := v_4 + v_1$
$v_6 := H(v_5)$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \right\}$$

- v_6 never even used in the program, clearly can be removed
- **Definition:** oracle constraint is **useless** if RHS is **linearly independent** of everything else

Normalize: useless oracle queries

Other kinds of oracle constraints can be removed, too!

<u>P:</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := v_1 + v_2$
$v_4 \leftarrow \mathbb{F}$
$v_5 := v_4 + v_1$
$v_6 \leftarrow \mathbb{F}$
return (v_4, v_5)

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \cancel{ \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \mapsto \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_4 & v_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{array} } \right\}$$

- v_6 never even used in the program, clearly can be removed
- **Definition:** oracle constraint is **useless** if RHS is **linearly independent** of everything else
- Useless oracle constraints can be removed, no effect on Adv

Subtleties about useless constraints

$$\begin{array}{l}\underline{\mathsf{P}}\\ v_1 \leftarrow \mathbb{F}\\ v_2 \leftarrow \mathbb{F}\\ v_3 := H(v_1)\\ v_4 := H(v_3)\\ v_5 = v_2 + v_4\\ \text{return } v_1, v_5\end{array}$$

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ [1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0], \quad [0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1] \Big\}$$

Subtleties:

Subtleties about useless constraints

$\underline{\mathsf{P}}$
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := H(v_1)$
$v_4 := H(v_3)$
$v_5 = v_2 + v_4$
return v_1, v_5

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ [1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0], \\ [0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1] \Big\}$$

Subtleties:

- v_4 *syntactically* influences the output, but still **useless**

Subtleties about useless constraints

$\underline{\mathsf{P}}$
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := H(v_1)$
$v_4 := H(v_3)$
$v_5 = v_2 + v_4$
return v_1, v_5

$$\mathcal{M} = \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathcal{C} = \Big\{ [1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0], \\ [0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1] \Big\}$$

Subtleties:

- v_4 *syntactically* influences the output, but still **useless**

Subtleties about useless constraints

<u>P</u>
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := H(v_1)$
$v_4 \leftarrow \mathbb{F}$
$v_5 = v_2 + v_4$
return v_1, v_5

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \Big\{ [1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0], \\ \xcancel{[0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1]} \Big\}$$

Subtleties:

- v_4 *syntactically* influences the output, but still **useless**

Subtleties about useless constraints

$\underline{\text{P}}$
$v_1 \leftarrow \mathbb{F}$
$v_2 \leftarrow \mathbb{F}$
$v_3 := H(v_1)$
$v_4 \leftarrow \mathbb{F}$
$v_5 = v_2 + v_4$
return v_1, v_5

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ [1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0], \cancel{[0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1]} \right\}$$

Subtleties:

- v_4 *syntactically* influences the output, but still **useless**
- Removing constraint #2 causes constraint #1 to be useless!

Subtleties about useless constraints

P

$v_1 \leftarrow \mathbb{F}$

$v_2 \leftarrow \mathbb{F}$

$v_3 \leftarrow \mathbb{F}$

$v_4 \leftarrow \mathbb{F}$

$v_5 = v_2 + v_4$

return v_1, v_5

$$\mathcal{M} = \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

$$\mathcal{C} = \left\{ \cancel{[1\ 0\ 0\ 0] \mapsto [0\ 0\ 1\ 0]}, \cancel{[0\ 0\ 1\ 0] \mapsto [0\ 0\ 0\ 1]} \right\}$$

Subtleties:

- v_4 *syntactically* influences the output, but still **useless**
- Removing constraint $\#2$ causes constraint $\#1$ to be useless!

Normalization

normalize(P):

iteratively delete **unreachable** and **useless** oracle constraints from P

Normalization

normalize(P):

iteratively delete **unreachable** and **useless** oracle constraints from P

Lemma: **normalize**$(P) \cong P$.

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Tools:

- **Algebraic representation** - output distribution of Linicrypt programs as matrices.
- **Normal form** - remove "extra" oracle queries.
- **Basis changes** - reorder the variables.

Basis changes

- We defined <u>reachable</u> & <u>useful</u> in terms of **linear independence**:
  - Constraint is <u>reachable</u> if LHS is in span of reachable vectors.
  - Constraint is <u>useful</u> if RHS is in span of reachable vectors.

Basis changes

- We defined reachable & useful in terms of **linear independence**:
  - Constraint is reachable if LHS is in span of reachable vectors.
  - Constraint is useful if RHS is in span of reachable vectors.
- Definitions are invariant under a **basis change** to all the vectors.

Basis changes

- We defined reachable & useful in terms of **linear independence**:
  - Constraint is reachable if LHS is in span of reachable vectors.
  - Constraint is useful if RHS is in span of reachable vectors.
- Definitions are invariant under a **basis change** to all the vectors.

Proposition: Linicrypt program $\mathbf{P}$ is **indistinguishable** from $\mathbf{B}\times\mathbf{P}$, where $\mathbf{B}$ is invertible matrix.

Linicrypt technical tools

Theorem: can decide, in polynomial time, whether two *input-less* Linicrypt programs have **indistinguishable output** distributions.

Tools:

- **Algebraic representation** - output distribution of Linicrypt programs as matrices.
- **Normal form** - remove “extra” oracle queries.
- **Basis changes** - reorder the variables.

Bringing it all together

Main Theorem:

$P_1 \cong P_2$ if and only if
normalize(P_1) and **normalize**(P_2) differ by a basis change.

Bringing it all together

Main Theorem:
$P_1 \cong P_2$ if and only if
normalize(P_1) and **normalize**(P_2) differ by a basis change.

Basic outline:

- Choose basis change for **normalize**(P_1) so that it coincides with **normalize**(P_2) "as much as possible"
- If they completely coincide, then $P_1 \cong P_2$

Bringing it all together

Main Theorem:
$P_1 \cong P_2$ if and only if
normalize(P_1) and **normalize**(P_2) differ by a basis change.

Basic outline:

- Choose basis change for **normalize**(P_1) so that it coincides with **normalize**(P_2) "as much as possible"
- If they completely coincide, then $P_1 \cong P_2$
- Otherwise, there is an oracle constraint that is ...
  - ... present in P_1 but not in P_2 (by symmetry)
  - ... **reachable** $\Rightarrow$ distinguisher can query it
  - ... **useful** $\Rightarrow$ result involved in some linear relation

  To distinguish, make this query, check that result satisfies the linear relation.

Outline

1. Linicrypt model & technical tools
2. **Synthesizing Linicrypt programs**
3. Applications to Garbled Circuits
4. Future work, open problems

Existential Formulas

Claim: Linicrypt properties can be expressed in an **existential formula.**

Existential Formulas

> **Claim:** Linicrypt properties can be expressed in an **existential formula.**

1. Are two Linicrypt programs indistinguishable?
2. Does the **composition** of two Linicrypt programs have a certain property

Trick: basis change witness

Problem: process for identifying **unreachable** oracle constraints is **highly iterative**.

Trick: basis change witness

Problem: process for identifying **unreachable** oracle constraints is **highly iterative**.

Reachable space is the smallest subspace $\mathcal{V}$ s.t.:

- $\mathcal{V}$ contains all rows of $\mathcal{M}$
- For all oracle constraints, if LHS in $\mathcal{V}$ then RHS in $\mathcal{V}$

Trick: basis change witness

Problem: process for identifying **unreachable** oracle constraints is **highly iterative**.

Reachable space is the smallest subspace $\mathcal{V}$ s.t.:

- $\mathcal{V}$ contains all rows of $\mathcal{M}$
- For all oracle constraints, if LHS in $\mathcal{V}$ then RHS in $\mathcal{V}$

Idea: guess a basis change under which $\mathcal{V} = \mathbb{F}^d \times \{0\}^{n-d}$.

- Guess basis B and its inverse. Check that $B \times B^{-1} = I$
- **A vector is in $\mathcal{V}$ $\Leftrightarrow$ rightmost $n - d$ entries are all zero.**

Trick: basis change witness

Problem: process for identifying **unreachable** oracle constraints is **highly iterative**.

Reachable space is the smallest subspace $\mathcal{V}$ s.t.:

- $\mathcal{V}$ contains all rows of $\mathcal{M}$
- For all oracle constraints, if LHS in $\mathcal{V}$ then RHS in $\mathcal{V}$

Idea: guess a basis change under which $\mathcal{V} = \mathbb{F}^d \times \{0\}^{n-d}$.

- Guess basis B and its inverse. Check that $B \times B^{-1} = I$
- **A vector is in $\mathcal{V}$ $\Leftrightarrow$ rightmost $n-d$ entries are all zero.**

Now easy to verify that we've identified the reachable space:

- After basis change, all rows of $\mathcal{M}$ in $\mathcal{V}$?
- After basis change, LHS of constraint in $\mathcal{V}$ $\Rightarrow$ RHS is too?

Trick: basis change for composition

Composition:

Use a **basis change** to “line up” base vars.

Composed output is second program’s output.

Trick: basis change for composition

Composition:
Use a **basis change** to "line up" base vars.
Composed output is second program's output.

$\underline{\mathrm{Enc}(m)\text{:}}$
$r \leftarrow \mathbb{F}$
return $(r, m + H(r))$

$$\rightarrow \begin{array}{ccc} m & r & H(r) \end{array} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$\underline{\mathrm{Dec}(r, c)\text{:}}$
return $c - H(r)$

$$\rightarrow \begin{array}{ccc} r & c & H(r) \end{array} \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \left\{ [1\ 0\ 0] \mapsto [0\ 0\ 1] \right\}$$

Trick: basis change for composition

Composition:

Use a **basis change** to "line up" base vars.

Composed output is second program's output.

$\underline{\mathrm{Enc}(m):}$

$r \leftarrow \mathbb{F}$

return $(r, m + H(r))$

$$\rightarrow \begin{array}{ccc} m & r & H(r) \end{array} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$\underline{\mathrm{Dec}(r, c):}$

return $c - H(r)$

$$\rightarrow \begin{array}{ccc} r & c & H(r) \end{array} \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \left\{ [1\ 0\ 0] \mapsto [0\ 0\ 1] \right\}$$

Trick: basis change for composition

Composition:
Use a **basis change** to "line up" base vars.
Composed output is second program's output.

$\underline{\mathrm{Enc}(m):}$
$r \leftarrow \mathbb{F}$
return $(r, m + H(r))$

$$\to \begin{bmatrix} \overset{m}{0} & \overset{r}{1} & \overset{H(r)}{0} \\ 1 & 0 & 1 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$\underline{\mathrm{Dec}(r, c):}$
return $c - H(r)$

$$\to \begin{bmatrix} \overset{r}{0} & \overset{c}{1} & \overset{H(r)}{-1} \end{bmatrix} \left\{ [1\ 0\ 0] \mapsto [0\ 0\ 1] \right\}$$

$$B = \begin{array}{cc} & \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{array}{c} r \\ c \\ H(r) \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Trick: basis change for composition

Composition:

Use a **basis change** to "line up" base vars.
Composed output is second program's output.

$\underline{\mathsf{Enc}(m)}$:
$r \leftarrow \mathbb{F}$
return $(r, m + H(r))$

$$\rightarrow \begin{array}{ccc} m & r & H(r) \\ \end{array}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$\underline{\mathsf{Dec}(r, m + H(r))}$:
return $(m + H(r)) - H(r)$

$$\rightarrow \begin{array}{ccc} m & r & H(r) \\ \end{array}\begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$$B = \begin{array}{c|ccc} & m & r & H(r) \\ \hline r & 0 & 1 & 0 \\ c & 1 & 0 & 1 \\ H(r) & 0 & 0 & 1 \end{array}$$

Trick: basis change for composition

Composition:

Use a **basis change** to "line up" base vars.
Composed output is second program's output.

$\underline{\mathrm{Enc}(m)}$:
$r \leftarrow \mathbb{F}$
return $(r, m + H(r))$

$$\rightarrow \begin{array}{c} \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \end{array} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$\underline{\mathrm{Dec}(r, m + H(r))}$:
return m

$$\rightarrow \begin{array}{c} \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \end{array} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$$B = \begin{array}{c} \\ r \\ c \\ H(r) \end{array} \begin{array}{c} \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Trick: basis change for composition

Composition:
Use a **basis change** to "line up" base vars.
Composed output is second program's output.

$\underline{\text{Dec}\circ\text{Enc}(m)\text{:}}$
return m

$$\rightarrow \begin{bmatrix} \scriptstyle m & \scriptstyle r & \scriptstyle H(r) \\ 1 & 0 & 0 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

$$B = \begin{array}{c c} & \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{array}{c} r \\ c \\ H(r) \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Trick: basis change for composition

Composition:
Use a **basis change** to "line up" base vars.
Composed output is second program's output.

Dec∘Enc(m):
return m

$$\rightarrow \begin{bmatrix} {\scriptstyle m} & {\scriptstyle r} & {\scriptstyle H(r)} \\ 1 & 0 & 0 \end{bmatrix} \left\{ [0\ 1\ 0] \mapsto [0\ 0\ 1] \right\}$$

Bonus: **B** is a witness to the **correctness** of this scheme!

$$B = \begin{array}{c c} & \begin{array}{ccc} m & r & H(r) \end{array} \\ \begin{array}{c} r \\ c \\ H(r) \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

Outline

1. Linicrypt model & technical tools
2. Synthesizing Linicrypt programs
3. **Applications to Garbled Circuits**
4. Future work, open problems

Garbled Circuits

Existing garbled-circuit constructions are Linicrypt*:

	cost per gate
Textbook GC [Yao80s]	4 field elements
[NaorPinkasSumner99]	3 field elements
[Pinkas+09]	2 field elements
[KolesnikovSchneider08]	XOR gate = 0; AND gate = 3
[ZahurREvans15]	XOR gate = 0; AND gate = 2

Garbled Circuits

Existing garbled-circuit constructions are Linicrypt*:

	cost per gate
Textbook GC [Yao80s]	4 field elements
[NaorPinkasSumner99]	3 field elements
[Pinkas+09]	2 field elements
[KolesnikovSchneider08]	XOR gate = 0; AND gate = 3
[ZahurREvans15]	XOR gate = 0; AND gate = 2

Can we use Linicrypt to **discover new** GC constructions?
prove **lower bounds** about GC constructions?

Garbled Circuits

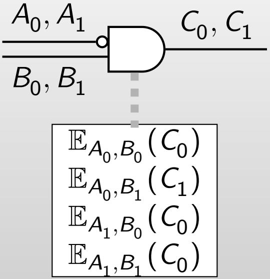
Existing garbled-circuit constructions are Linicrypt*:

	cost per gate
Textbook GC [Yao80s]	4 field elements
[NaorPinkasSumner99]	3 field elements
[Pinkas+09]	2 field elements
[KolesnikovSchneider08]	XOR gate = 0; AND gate = 3
[ZahurREvans15]	XOR gate = 0; AND gate = 2

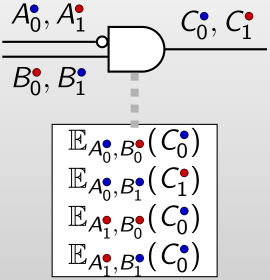
Can we use Linicrypt to **discover new** GC constructions?
prove **lower bounds** about GC constructions?

(focus on constructions to garble small **gates**)

Point/permute [BeaverMicaliRogaway90]

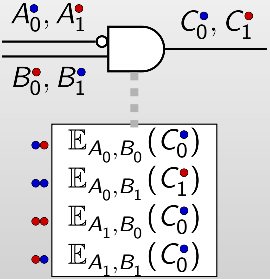


Point/permute [BeaverMicaliRogaway90]



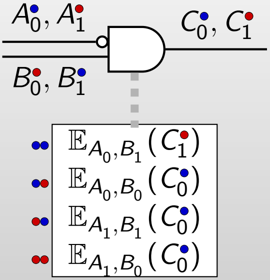
- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)

Point/permute [BeaverMicaliRogaway90]



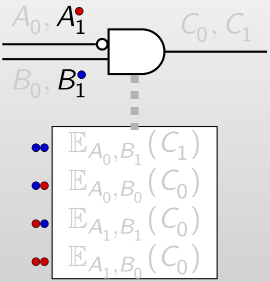
- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point/permute [BeaverMicaliRogaway90]



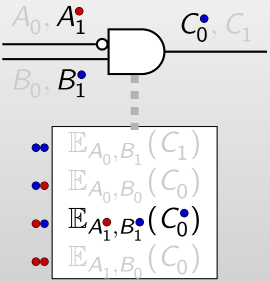
- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys

Point/permute [BeaverMicaliRogaway90]



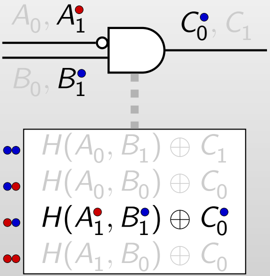
- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Point/permute [BeaverMicaliRogaway90]



- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

Point/permute [BeaverMicaliRogaway90]

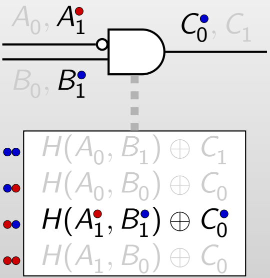


- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

✔ $\mathbb{E}_{A,B}(C)$ doesn't need verifiable decryption

$\Rightarrow$ smaller garbled circuit, encryption/decryption "linear"

Point/permute [BeaverMicaliRogaway90]



- Randomly assign (•, •) or (•, •) to each pair of wire labels
- Include color in the wire label (e.g., as last bit)
- Order the 4 ciphertexts canonically, by color of keys
- Evaluate by decrypting ciphertext indexed by your colors

✔ $\mathbb{E}_{A,B}(C)$ doesn't need verifiable decryption

$\Rightarrow$ smaller garbled circuit, encryption/decryption "linear"

✗ Evaluator's **choice of linear operation** depends on color bit in a non-linear way!

Modeling Garbled Circuits in Linicrypt

- Garbler's behavior is Linicrypt program, after fixing **association between T/F and** $\bullet/\bullet$ on both wires
- Evaluator's behavior is Linicrypt program, after fixing **color bits of input wire labels**

Modeling Garbled Circuits in Linicrypt

- Garbler's behavior is Linicrypt program, after fixing **association between T/F and** ${\color{blue}\bullet}/{\color{red}\bullet}$ on both wires
- Evaluator's behavior is Linicrypt program, after fixing **color bits of input wire labels**

Our approach: Synthesize collection of Linicrypt programs:

$$\{\mathsf{Garble}^{{\color{red}\bullet\bullet}}, \mathsf{Garble}^{{\color{red}\bullet}{\color{blue}\bullet}}, \mathsf{Garble}^{{\color{blue}\bullet}{\color{red}\bullet}}, \mathsf{Garble}^{{\color{blue}\bullet\bullet}}\}$$

$$\{\mathsf{Eval}^{{\color{red}\bullet\bullet}}, \mathsf{Eval}^{{\color{red}\bullet}{\color{blue}\bullet}}, \mathsf{Eval}^{{\color{blue}\bullet}{\color{red}\bullet}}, \mathsf{Eval}^{{\color{blue}\bullet\bullet}}\}$$

. . . that comprise a point-permute-style GC construction

Modeling Garbled Circuits in Linicrypt

- Garbler's behavior is Linicrypt program, after fixing **association between T/F and** $\textcolor{blue}{\bullet}/\textcolor{red}{\bullet}$ on both wires
- Evaluator's behavior is Linicrypt program, after fixing **color bits of input wire labels**

Our approach: Synthesize collection of Linicrypt programs:

$\{\mathsf{Garble}^{\textcolor{red}{\bullet}\textcolor{red}{\bullet}}, \mathsf{Garble}^{\textcolor{red}{\bullet}\textcolor{blue}{\bullet}}, \mathsf{Garble}^{\textcolor{blue}{\bullet}\textcolor{red}{\bullet}}, \mathsf{Garble}^{\textcolor{blue}{\bullet}\textcolor{blue}{\bullet}}\}$

$\{\mathsf{Eval}^{\textcolor{red}{\bullet}\textcolor{red}{\bullet}}, \mathsf{Eval}^{\textcolor{red}{\bullet}\textcolor{blue}{\bullet}}, \mathsf{Eval}^{\textcolor{blue}{\bullet}\textcolor{red}{\bullet}}, \mathsf{Eval}^{\textcolor{blue}{\bullet}\textcolor{blue}{\bullet}}\}$

... that comprise a point-permute-style GC construction

Details I won't discuss:

- GC security $\Leftrightarrow$ indistinguishability of input-less Linicrypt program
- How to express correctness of GC scheme via composition
- We specialized to Free-XOR-compatible schemes

Linisynth : `github.com/osu-crypto/linisynth`

Linisynth architecture

Linisynth : `github.com/osu-crypto/linisynth`

Linisynth architecture

1. Get **parameters** as input: number of oracle queries allowed, size (# field elts) of garbled gate, gate functionality, etc.

Linisynth : `github.com/osu-crypto/linisynth`

Linisynth architecture

1. Get **parameters** as input: number of oracle queries allowed, size (# field elts) of garbled gate, gate functionality, etc.
2. For each **Garble**, **Eval** program, create existential variables for its matrix/constraint representation.

Linisynth : `github.com/osu-crypto/linisynth`

Linisynth architecture

1. Get **parameters** as input: number of oracle queries allowed, size (# field elts) of garbled gate, gate functionality, etc.
2. For each **Garble**, **Eval** program, create existential variables for its matrix/constraint representation.
3. Generate formula expressing security & correctness.

Linisynth : `github.com/osu-crypto/linisynth`

Linisynth architecture

1. Get **parameters** as input: number of oracle queries allowed, size (# field elts) of garbled gate, gate functionality, etc.
2. For each **Garble**, **Eval** program, create existential variables for its matrix/constraint representation.
3. Generate formula expressing security & correctness.
4. Ask **Z3** whether the formula is satisfiable.

Linisynth : github.com/osu-crypto/linisynth

Linisynth architecture

1. Get **parameters** as input: number of oracle queries allowed, size (# field elts) of garbled gate, gate functionality, etc.
2. For each **Garble**, **Eval** program, create existential variables for its matrix/constraint representation.
3. Generate formula expressing security & correctness.
4. Ask **Z3** whether the formula is satisfiable.

Results

name	τ	size	H_{gb}	H_{ev}	time	sat
free-xor	$\oplus : 2 \to 1$	0	0	0	1s	1
half-gate	$\wedge : 2 \to 1$	2	4	2	5s	1
one-third-gate	$\wedge : 2 \to 1$	1	4	2	74s	0
half-gate-cheaper	$\wedge : 2 \to 1$	2	4	1	6.2h	0
1-out-of-2-mux	$\text{MUX} : 3 \to 1$	2	4	2	29s	1
2-bit-eq	$= : 4 \to 1$	2	4	2	6m	1
2-bit-eq-small	$= : 4 \to 1$	1	4	2	6m	0
2-bit-leq	$\leq : 4 \to 1$	1	2	1	77s	0
2-bit-lt	$< : 4 \to 1$	2	4	2	3.5h	0

half-gate size $= 2$ calls$_{\mathsf{gb}} = 4$

$\wedge : \{0,1\}^2 \to \{0,1\}$ time $= 5\mathsf{s}$ calls$_{\mathsf{ev}} = 2$

$\underline{\mathsf{GateGb}^H(\sigma, A, B, \Delta):}$

$$
\begin{aligned}
&h_1 = H(A) \\
&h_2 = H(A + \Delta) \\
&h_3 = H(A + B) \\
&h_4 = H(A + B + \Delta) \\
&G_0 = [0,2]\Delta + h_3 + h_4 \\
&G_1 = A + B + [0,2]\Delta + h_1 + h_2 + h_3 + h_4 \\
&C_0 = B + [0]\Delta + [0,2]h_1 + [1,3]h_2 + [1,2]h_3 + [0,3]h_4 \\
&\text{return } G_0, G_1, C_0
\end{aligned}
$$

$\underline{\mathsf{GateEv}^H(\chi, A^*, B^*, G_0, G_1):}$

$$
\begin{aligned}
\text{return } &[1,3]A^* + [0,2]B^* + \\
&[0,1]G_0 + [1,3]G_1 + \\
&H(A^*) + H(A^* + B^*)
\end{aligned}
$$

Lower Bounds

Existential formula is satisfiable **if and only if**
a Linicrypt construction exists with those parameters

Lower Bounds

Existential formula is satisfiable **if and only if**
a Linicrypt construction exists with those parameters

SAT solver returns $\textsc{false} \Rightarrow$ **lower bound** for garbled circuit constructions in Linicrypt

Outline

1. Linicrypt model & technical tools
2. Synthesizing Linicrypt programs
3. Applications to Garbled Circuits
4. **Future work, open problems**

Future Work

Theoretical questions about the model:

1. Security properties involving adversarial inputs (vs. input-less)
2. Support "fancier" random oracles (ideal cipher)
3. Which Linicrypt programs have security from standard assumptions (vs. random oracle)?
4. Better understanding of concrete (vs. asymptotic) bounds. E.g., "beyond birthday" security

Future Work

Theoretical questions about the model:

1. Security properties involving adversarial inputs (vs. input-less)
2. Support "fancier" random oracles (ideal cipher)
3. Which Linicrypt programs have security from standard assumptions (vs. random oracle)?
4. Better understanding of concrete (vs. asymptotic) bounds. E.g., "beyond birthday" security

Application areas (synthesis & lower bounds):

1. Block cipher modes, tweakable block ciphers
2. Authenticated encryption modes, MACs
3. Hash-based signatures
4. More garbled circuits
5. Memory-hard functions??

Future Work

Theoretical questions about the model:

1. Security properties involving adversarial inputs (vs. input-less)
2. Support "fancier" random oracles (ideal cipher)
3. Which Linicrypt programs have security from standard assumptions (vs. random oracle)?
4. Better understanding of concrete (vs. asymptotic) bounds. E.g., "beyond birthday" security

Application areas (synthesis & lower bounds):

1. Block cipher modes, tweakable block ciphers
2. Authenticated encryption modes, MACs
3. Hash-based signatures
4. More garbled circuits
5. Memory-hard functions??

Thanks!