White Box Crypto

Avik Chakraborti, ISI Kolkata

Outline of the Talk

1 Intro

2 Obfuscation

3 White Box Crypto

4 White Box AES

Cryptographic Key Protection

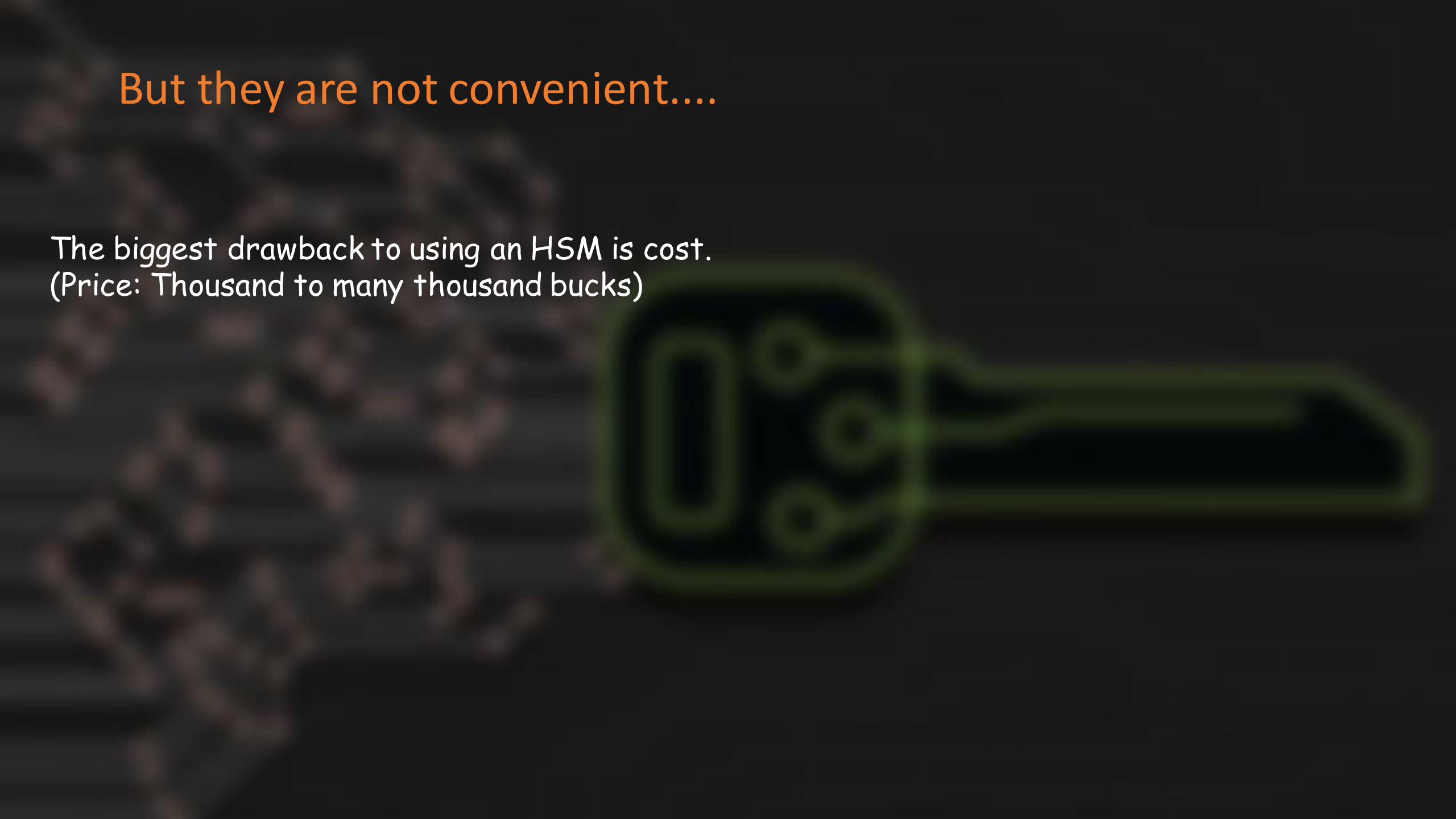
Hardware Security Model

Smart Card

Others.....

But they are not convenient....

The biggest drawback to using an HSM is cost.
(Price: Thousand to many thousand bucks)



But they are not convenient....

The biggest drawback to using an HSM is cost.
(Price: Thousand to many thousand bucks)

Difficulty in upgrading. If a weakness is exposed
it is not easy to upgrade. Bugs, security flaws might occur.

But they are not convenient....

The biggest drawback to using an HSM is cost.
(Price: Thousand to many thousand bucks)

Difficulty in upgrading. If a weakness is exposed
it is not easy to upgrade. Bugs, security flaws might occur.

ROCA Vulnerability, 2017

COMPLETELY BROKEN —

Millions of high-security crypto keys crippled by newly discovered flaw

Factorization weakness lets attackers impersonate key holders and decrypt their data.

DAN GOODIN (US) - 16/10/2017, 16:30

A bug in the firmware's prime-search algorithm used for RSA key generation results in RSA keys that are relatively cheap and inexpensive to factor. The bug impacts Infineon Trusted Platform Modules (TPMs) as well as many smartcards and Hardware Security Modules (HSMs) that use Infineon chips.

But they are not convenient....

The biggest drawback to using an HSM is cost.
(Price: Thousand to many thousand bucks)

Difficulty in upgrading. If a weakness is exposed
it is not easy to upgrade. Bugs, security flaws might occur.

Overall long lifecycle but limited updates

ROCA Vulnerability, 2017

A bug in the firmware's prime-search algorithm used for RSA key generation results in RSA keys that are relatively cheap and inexpensive to factor. The bug impacts Infineon Trusted Platform Modules (TPMs) as well as many smartcards and Hardware Security Modules (HSMs) that use Infineon chips.

Pure Software Based Apps

- Cheaper, lesser testing, easier to update
- Mobile Payment (HCE based):
  - No hardware Secure Element (SE)
  - Cloud Based Payment
  - Imitate Secure Element
- Digital Right Management
- IoT Application without SE
- Others . . . . . .

Protecting Keys

If Attacker has access to the implementation code (binary executable)

- Analyze the code
- Find the memory access instructions
- Filter those instructions involving key bits
- Scan memory for secret keys

Protecting Keys

If Attacker has access to the implementation code (binary executable)

- Analyze the code
- Find the memory access instructions
- Filter those instructions involving key bits
- Scan memory for secret keys

Adi Shamir and Van Someron: Find cryptographic keys from GigaBytes of datastream (Financial Crypto, 98)
Observation: Key information (in the middle of the figure) looks more noisy than the rest of the data

Protecting Keys

If Attacker has access to the implementation code (binary executable)

- Analyze the code
- Find the memory access instructions
- Filter those instructions involving key bits
- Scan memory for secret keys

Adi Shamir and Van Someron: Find cryptographic keys from GigaBytes of datastream (Financial Crypto, 98)
Observation: Key information (in the middle of the figure) looks more noisy than the rest of the data

Obfuscate the Code

Outline of the Talk

1 Intro

2 Obfuscation

3 White Box Crypto

4 White Box AES

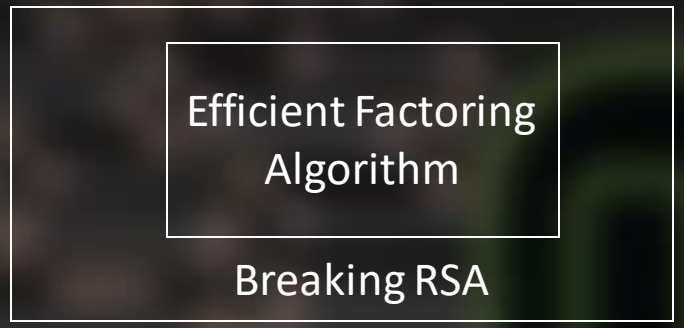
Obfuscation

Obfuscation is a function/act of obfuscating a code that is difficult for humans to understand

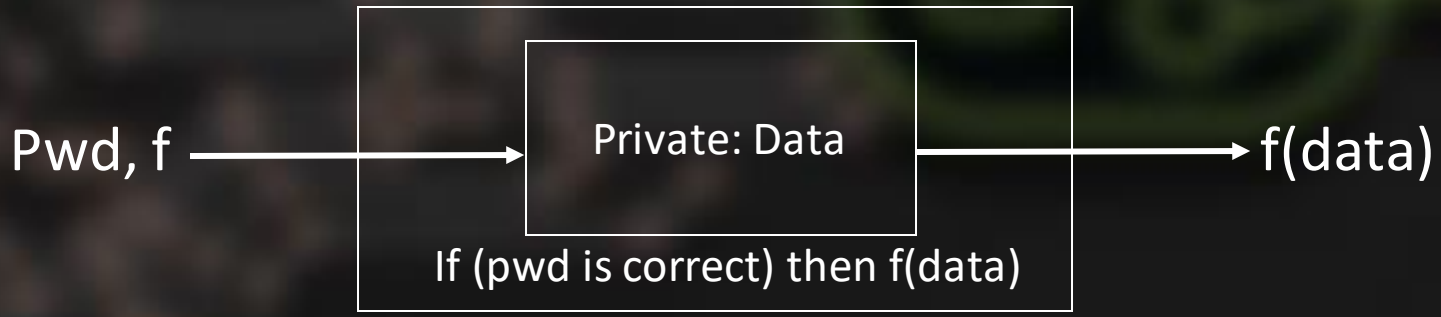
Obfuscation makes a program unintelligible while preserving its functionality

Why Obfuscation?

- To protect some secret inside a program
  - Secret algorithm or key (e.g. a factoring program)



  - some private data used by the program (e.g. conditional data access)



- Obfuscating "Hello World" program is useless

What is Program?

Few Points

- A word in language P $\in L$ **(some encoded string)**
- *execute* : $L \times D \rightarrow R$ s.t *out* = *execute* (P, *in*)
- P implements f: D → R, if for all $a \in A$, *execute* (P, *a*) = *f*(*a*), we say P ≡ *f*
- P1 and P2 are functionally equivalent if for some *f*, P1 ≡ *f* ≡ **P2,**
  we denote it P1 ≡ **P2**

Obfuscation

Obfuscator

- An algorithm O with input P, such that
- Functionality: $O(P) \equiv P$
- Efficiency: $O(P)$ is efficiently executable (n to poly (n))
- Security: → Next Slides

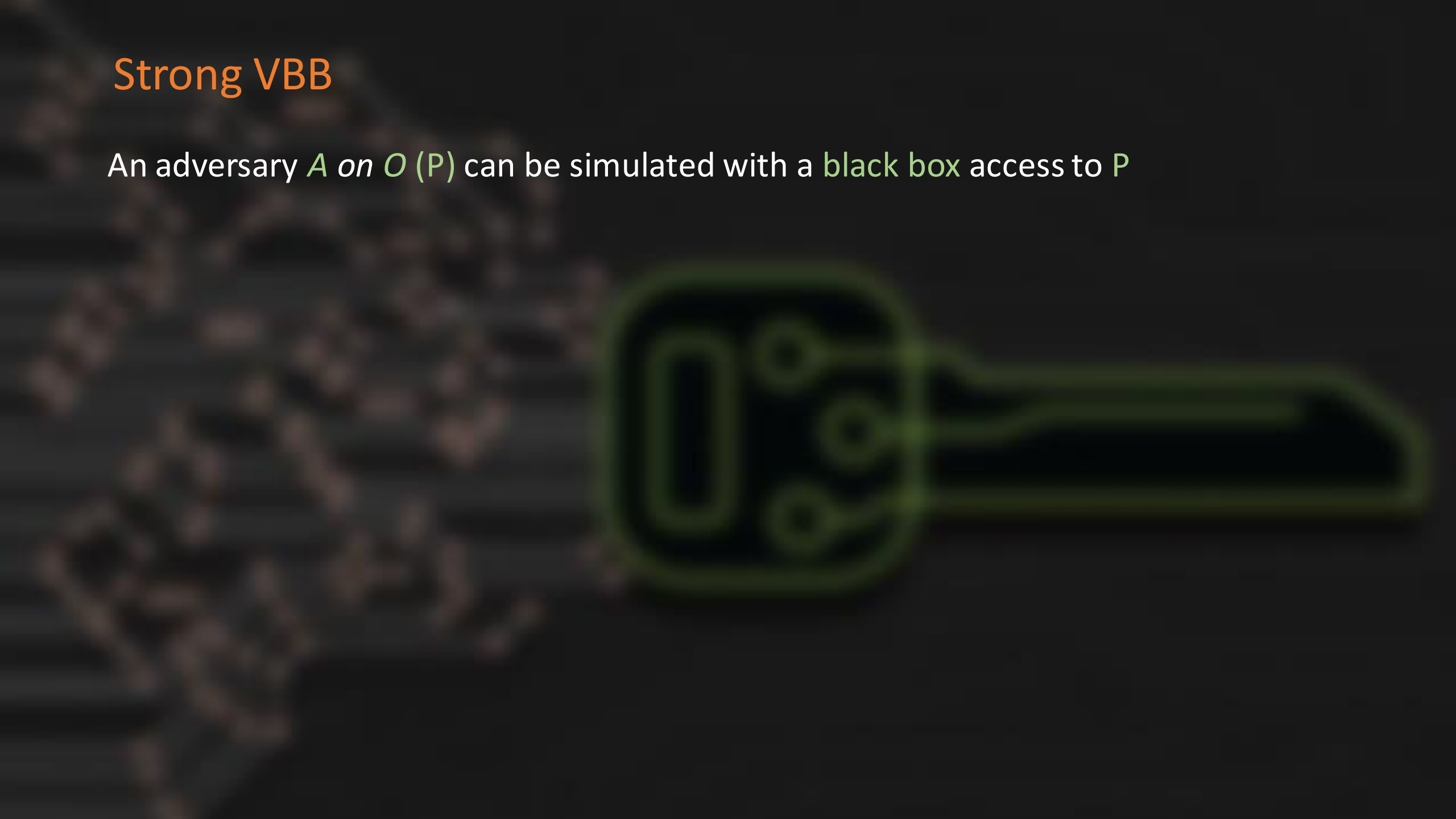
Virtual Black Box (VBB) Security

Interpretation

- *O* is a VBB obfuscator in the sense that anything one can efficiently compute given *O* (P), one could also effciently compute given Black Box oracle access to P.

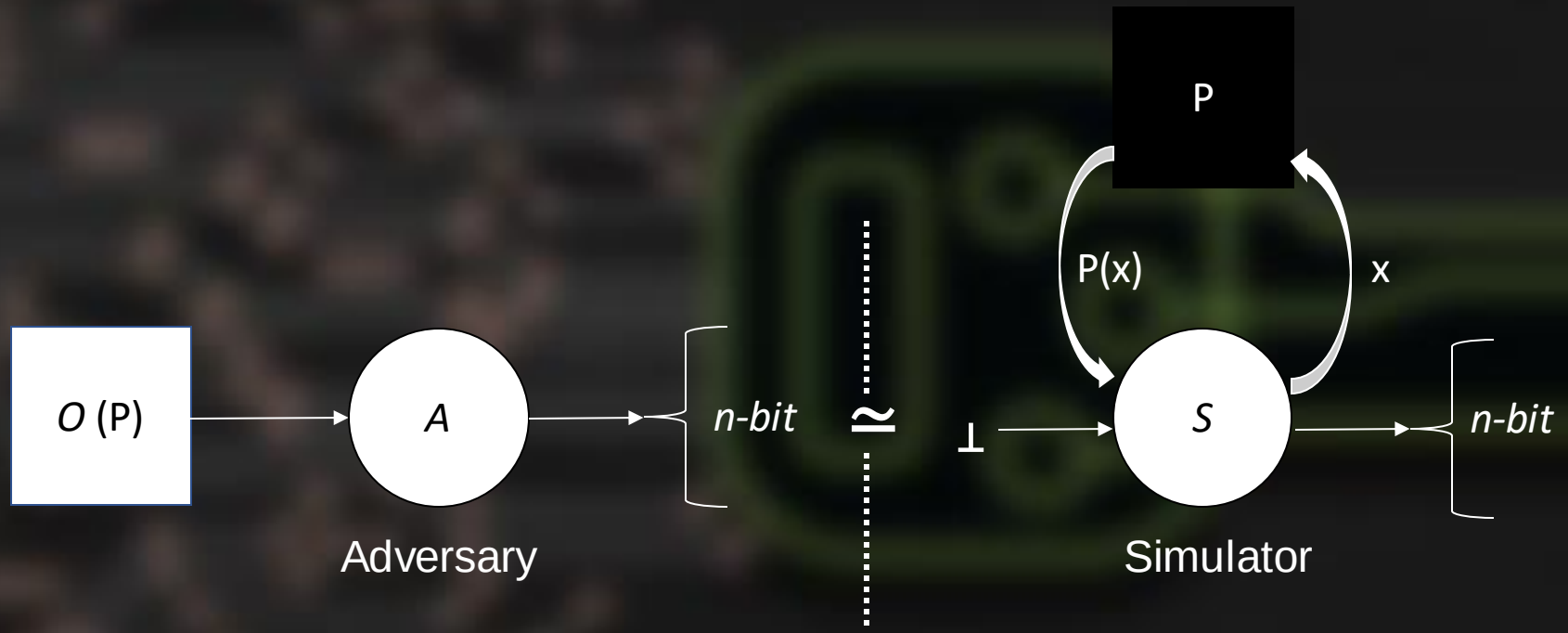
Strong VBB

An adversary *A* *on* *O* (P) can be simulated with a black box access to P



Strong VBB

An adversary *A* on *O* (P) can be simulated with a black box access to P



$$\{A(O(P))\}_n \simeq \{S^P(\perp)\}_n$$

Strong VBB

Too Strong

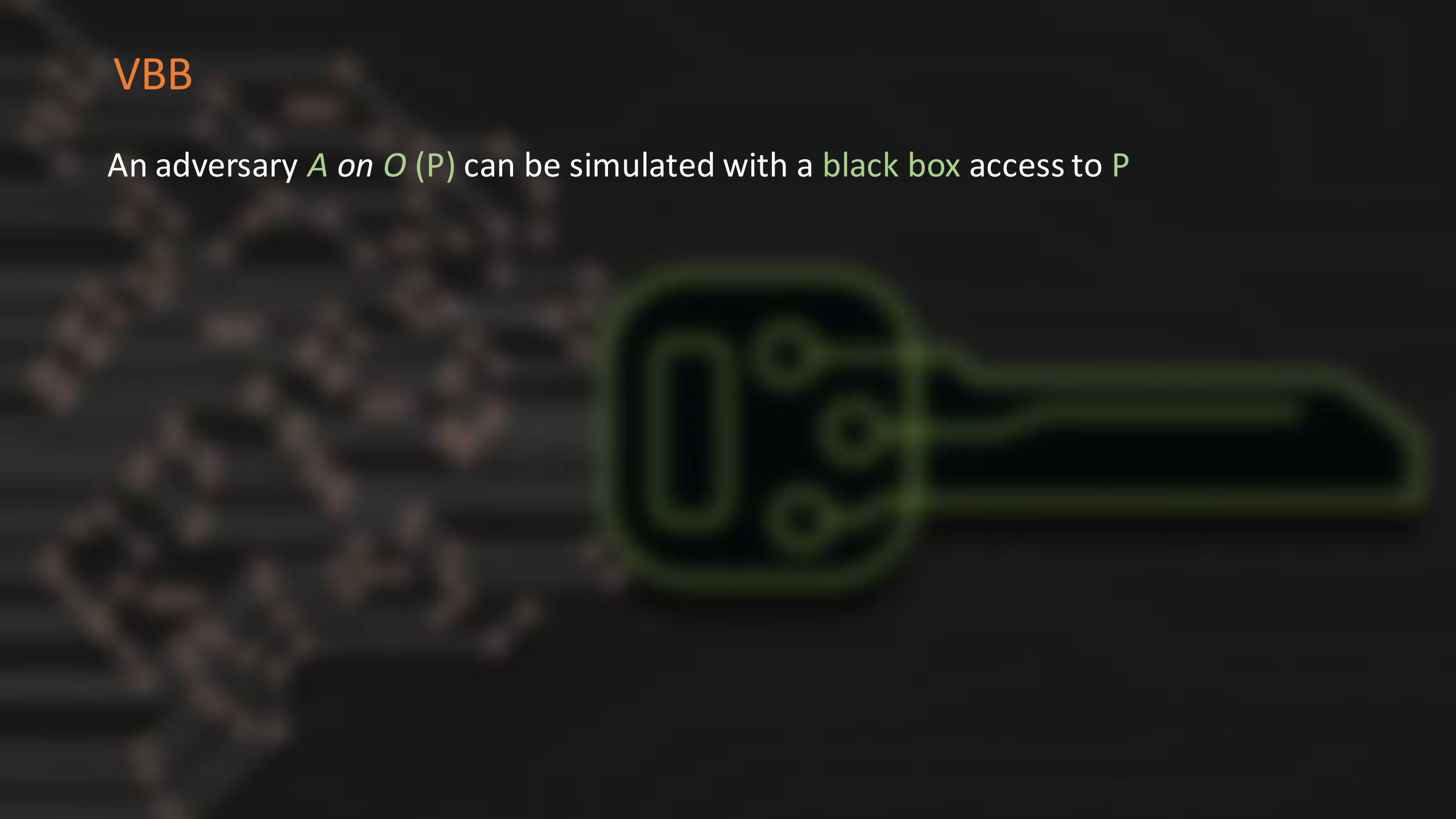
- Consider an adversary *A*, *such that* *A*(*O*(P)) outputs *O*(P)

- *S* has black-box access to P and it is really hard to simulate *O*(P), as *S* knows only several input / output pairs corresponding to its queries to P

- Weaker: Reduce the output length of *A* and *S*. Let *A* and *S* are *predicates*

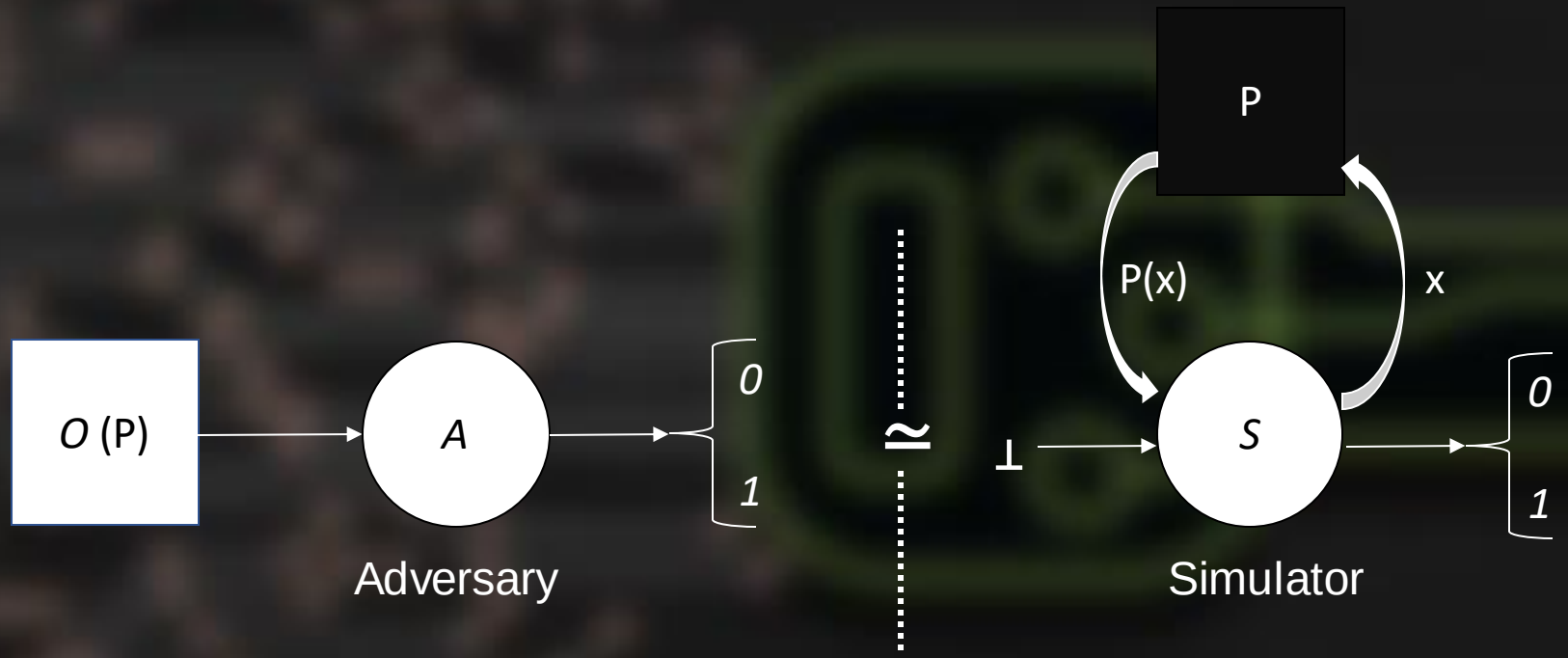
VBB

An adversary *A* *on* *O* (P) can be simulated with a black box access to P



VBB

An adversary *A* on *O* (P) can be simulated with a black box access to P



$$\{A(O(P))\}_1 \simeq \{S^P(\perp)\}_1$$

Impossibility of VBB (Barak et al., CRYPTO, 2001)

- VBB does not exist on general programs
- Idea: Construct a family of functions *F* which is *Inherently Unobfuscatable*, meaning there is property $\pi: F \rightarrow \{0, 1\}$, *such that*

  - Given any P computing $f \in F$, $\pi(f)$ can be computed efficiently
    (Note: $O(P)$ also computes f)

  - Given oracle access to a randomly selected $f \in F$, $\pi(f)$ can not be computed better than random guessing

Impossibility of VBB (Barak et al., CRYPTO, 2001)

"self-eating" programs

```
uint128_t cannibal ( prog P, uint128_t password )
{
    uint128_t secret1 = 0xe075b4f4eabf4377c1aa7202c8cc1ccb ;
    uint128_t secret2 = 0x94ff8ec818de3bd8223a62e4cb7c84a4 ;

    if ( password == secret1 ) return secret2 ;
    if ( execute (P, ⊥ , secret1 ) == secret2 ) return secret1 ;

    return 0;
}
```

Impossibility of VBB (Barak et al., CRYPTO, 2001)

"self-eating" programs

```
uint128_t cannibal ( prog P, uint128_t password )
{
    uint128_t secret1 = 0xe075b4f4eabf4377c1aa7202c8cc1ccb ;
    uint128_t secret2 = 0x94ff8ec818de3bd8223a62e4cb7c84a4 ;

    if ( password == secret1 ) return secret2 ;
    if ( execute (P, ⊥ , secret1 ) == secret2 ) return secret1 ;

    return 0;
}
```

$O(\text{cannibal})\ (O(\text{cannibal}); 0) = \text{secret1}$

Impossibility of VBB (Barak et al., CRYPTO, 2001)

"self-eating" programs

```
uint128_t cannibal ( prog P, uint128_t password )
{
    uint128_t secret1 = 0xe075b4f4eabf4377c1aa7202c8cc1ccb ;
    uint128_t secret2 = 0x94ff8ec818de3bd8223a62e4cb7c84a4 ;

    if ( password == secret1 ) return secret2 ;
    if ( execute (P, ⊥ , secret1 ) == secret2 ) return secret1 ;

    return 0;
}
```

(0 == secret1) is False

execute (*O*(cannibal), ⊥, **secret1) will** return secret2

O(cannibal) (*O*(cannibal); 0) = secret1

Impossibility of VBB (Barak et al., CRYPTO, 2001)

"self-eating" programs

```
uint128_t cannibal ( prog P, uint128_t password )
{
    uint128_t secret1 = 0xe075b4f4eabf4377c1aa7202c8cc1ccb ;
    uint128_t secret2 = 0x94ff8ec818de3bd8223a62e4cb7c84a4 ;

    if ( password == secret1 ) return secret2 ;
    if ( execute (P, ⊥ , secret1 ) == secret2 ) return secret1 ;

    return 0;
}
```

(0 == secret1) is False

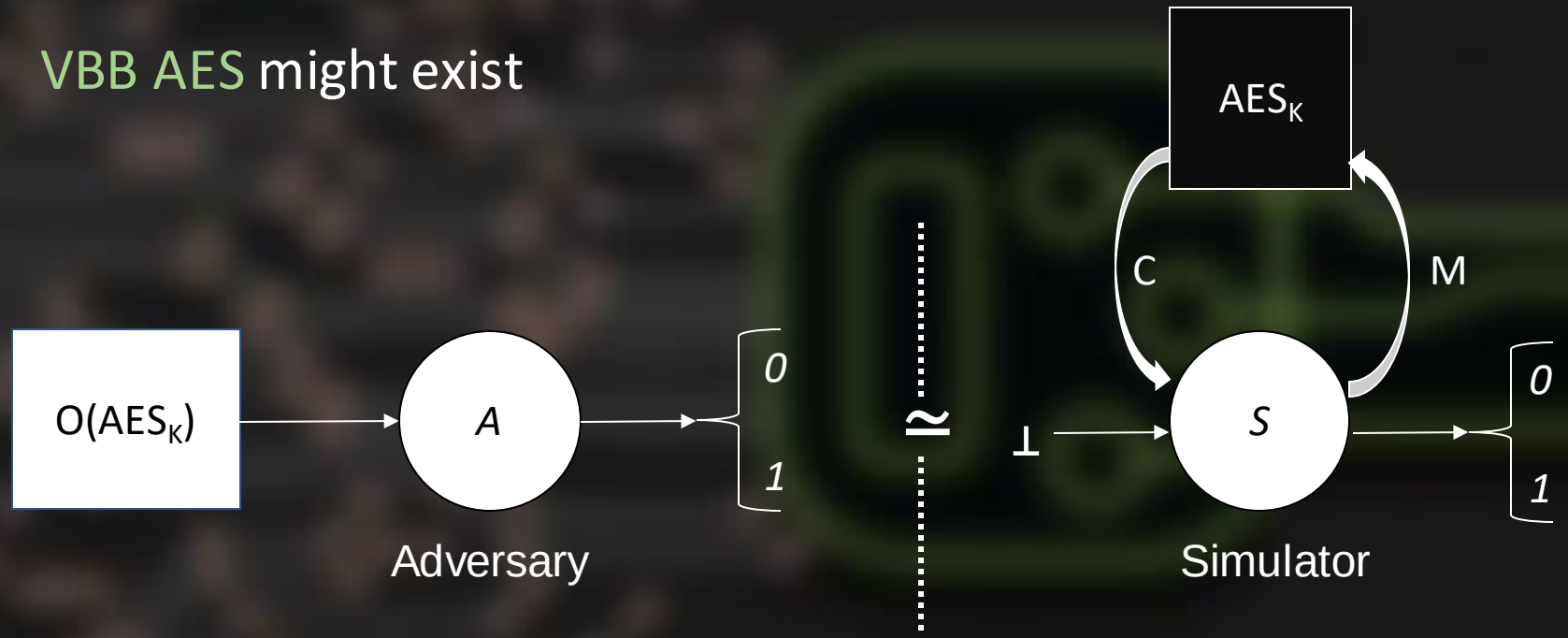
execute (*O*(cannibal), ⊥, **secret1) will** return secret2

O(cannibal) (*O*(cannibal); 0) = secret1

"A generic obfuscator does not exist, i.e., there exist programs that cannot be VBB-obfuscated."

The Good News

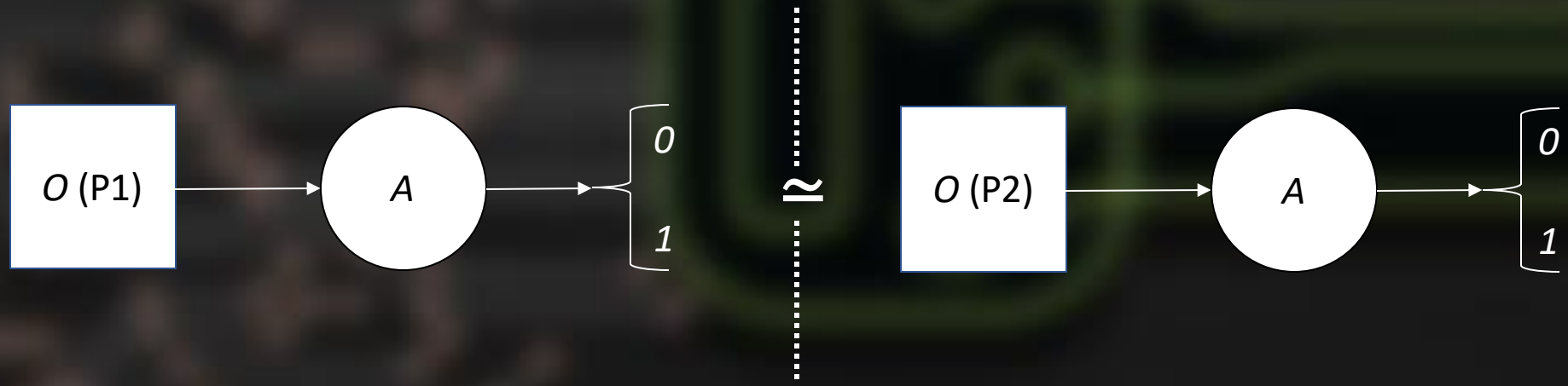
- The impossibility result does not apply to a specific encryption algorithm
- VBB AES might exist



- Bad News: Hard to achieve

Indistinguishability Obfuscation (IO), a Reasonably Weak Notion

- General purpose obfuscation is impossible
- Weaker Notion but covers a lot: IO
- For any P1 and P2, P1 ≡ P2 , implies O(P1) and O(P2) are indistinguishable



$$\{A(O(P1))\}_1 \simeq \{A(O(P2))\}_1$$

Outline of the Talk

1 Intro

2 Obfuscation

3 White Box Crypto

4 White Box AES

What is White Box Crypto (WBC)?

- “the attacker is assumed to have full access to the encrypting software and control of the execution environment”

- “Our main goal is to make key extraction difficult.”

– Chow et al. (DRM 2002)

What is White Box Crypto (WBC)?

- “the attacker is assumed to have full access to the encrypting software and control of the execution environment”

  ⇒ obfuscation restricted to encryption (or another crypto primitive)

- “Our main goal is to make key extraction difficult.”

– Chow et al. (DRM 2002)

What is White Box Crypto (WBC)?

- “the attacker is assumed to have full access to the encrypting software and control of the execution environment”

  ⇒ obfuscation restricted to encryption (or another crypto primitive)

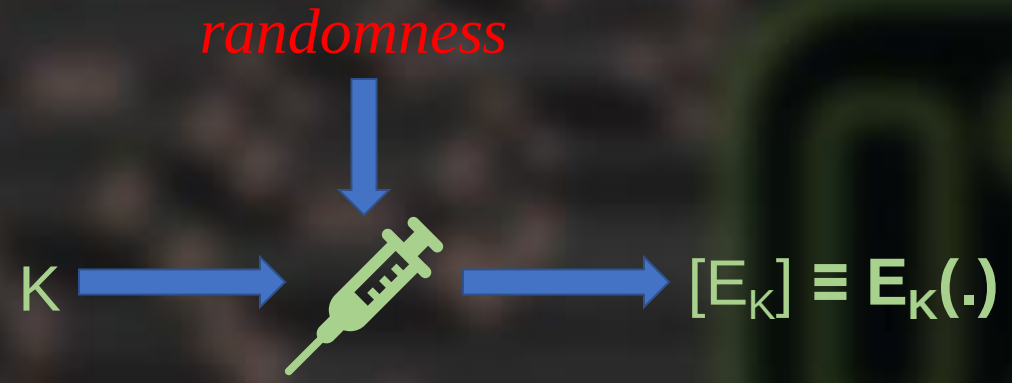
- “Our main goal is to make key extraction difficult.”

  ⇒ relaxed security requirements

– Chow et al. (DRM 2002)

What is White Box Crypto (WBC)?

- White-box cryptography is obfuscation of crypto code

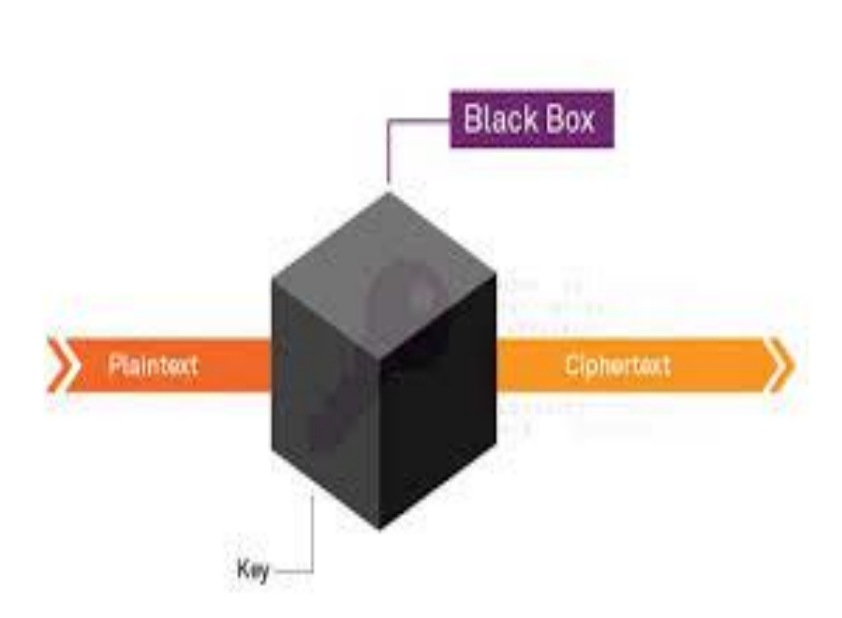


- It is an obfuscator that only resists Key Extraction

Why the Term "White Box"?

Black Box

- Zero visibility to code
- Visibility to external info
- Secure if the cipher has no weakness

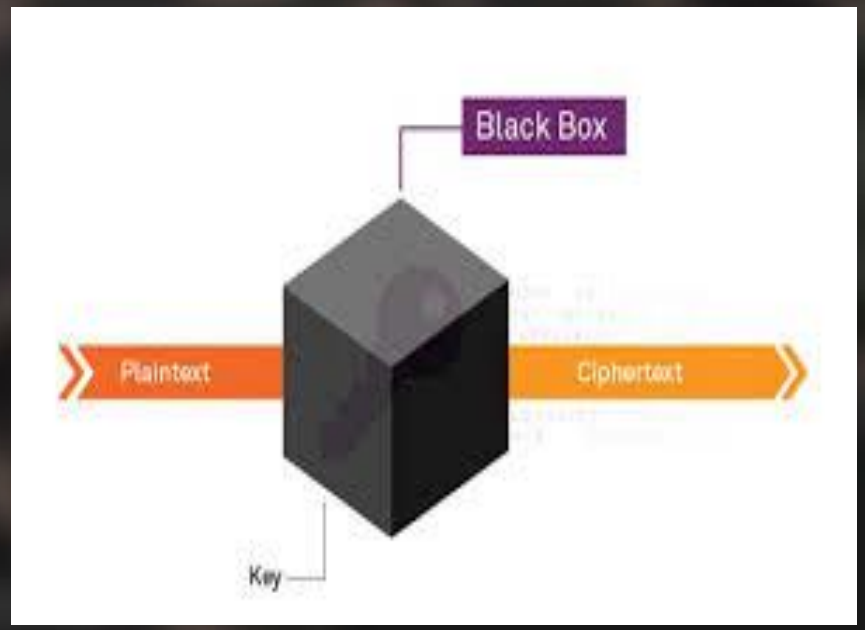


https://www.slideshare.net/Iqrakhalil2/white-box-crytography-in-an-insecure-enviroment

Why the Term "White Box"?

Black Box

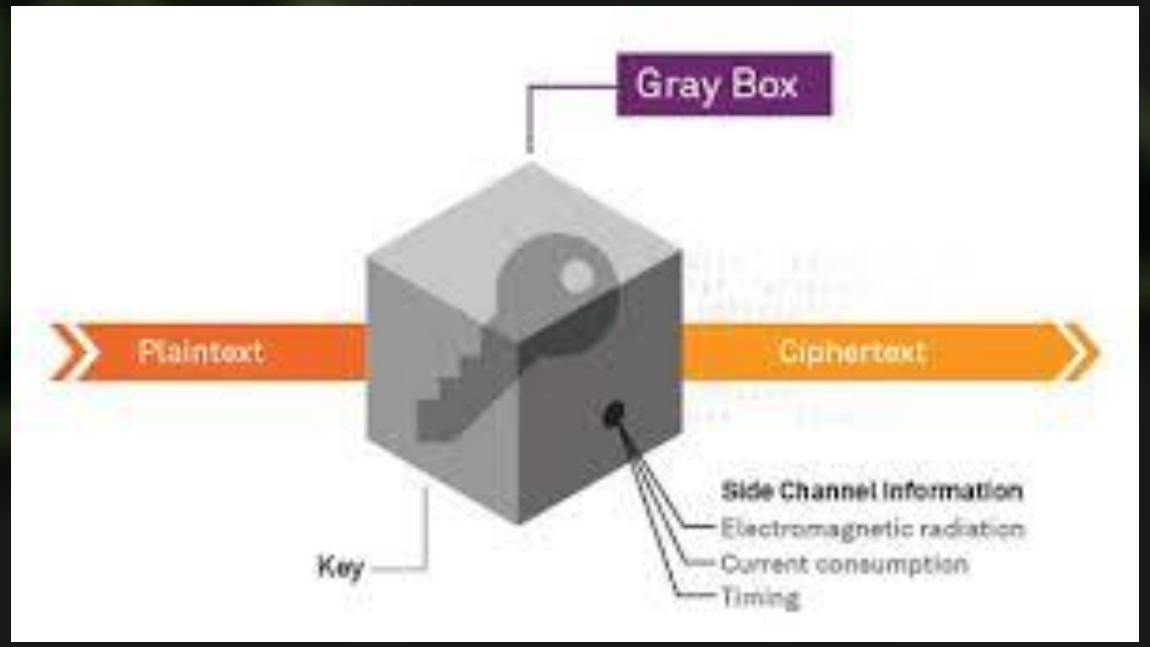
- Zero visibility to code
- Visibility to external info
- Secure if the cipher has no weakness



https://www.slideshare.net/Iqrakhalil2/white-box-crytography-in-an-insecure-enviroment

Gray Box

- Partial physical access to key due to information leakage
  Such as EM radiation, power consumption….

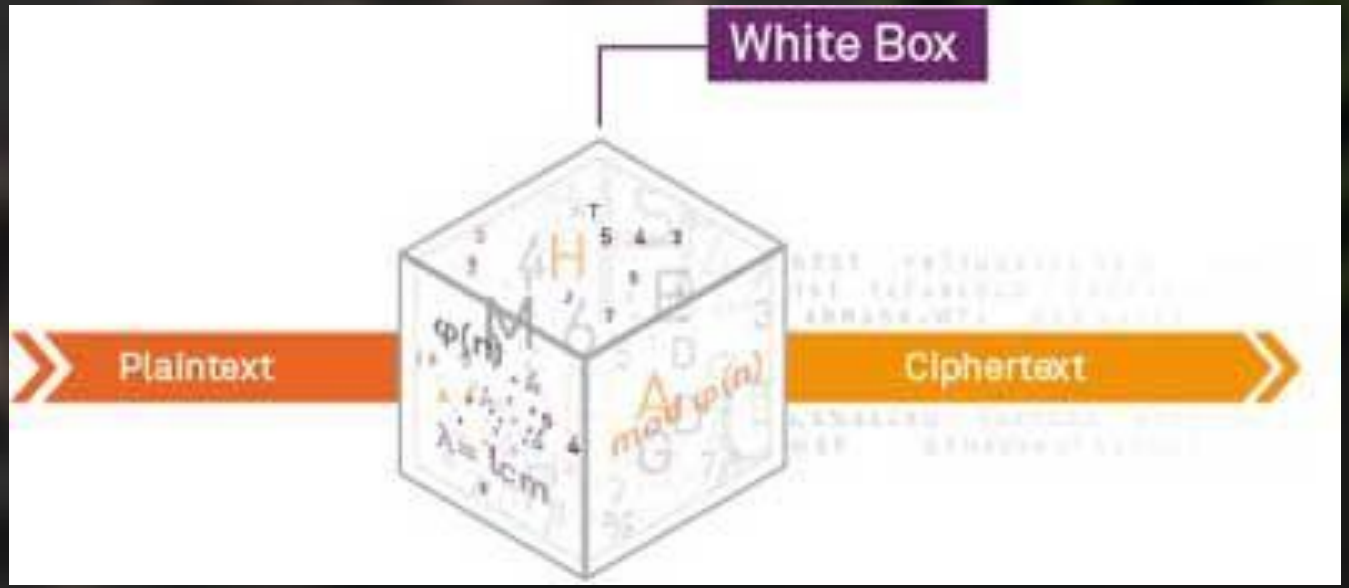


http://ijiet.com/wp-content/uploads/2017/05/31.pdf

Why the Term "White Box"?

White Box

- Full visibility to code
- Input, output, memory, internal computations
- Traditional Crypto is not secure under white box visibility



https://semiengineering.com/26374-2/

Strongest possible WBC

VBB obfuscation restricted to AES

- Impossibility result does not apply
- The AES-LUT program achieves VBB : $10^9 \cdot 10^9 \cdot 10^9$ TB
- Compact VBB-AES could be impossible

Weaker WBC

- Is this a Good Obfuscator ?

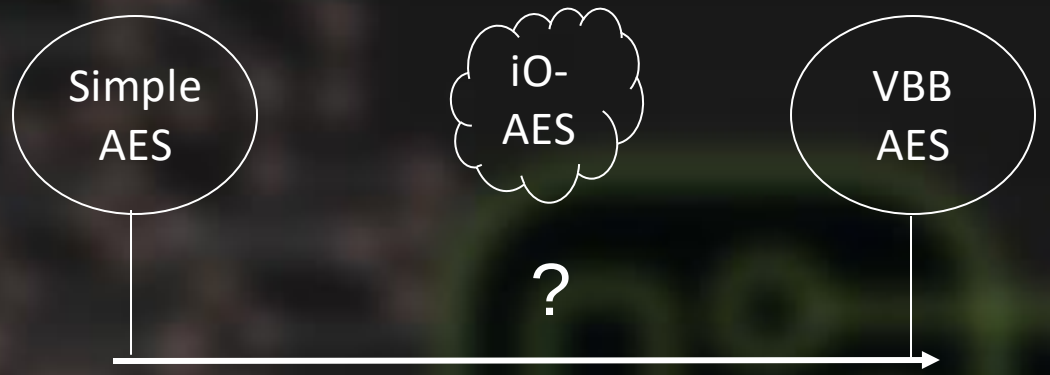
1. k <- Extract-key(P_k)
2. return reference implementation AES_k

- This is an iO AES

- Good If key-extraction is difficult on some P_k
  - key-extraction is still difficult on $O(P_k)$
  - key-extraction is difficult on any $O(P_k) \simeq O(P'_k)$

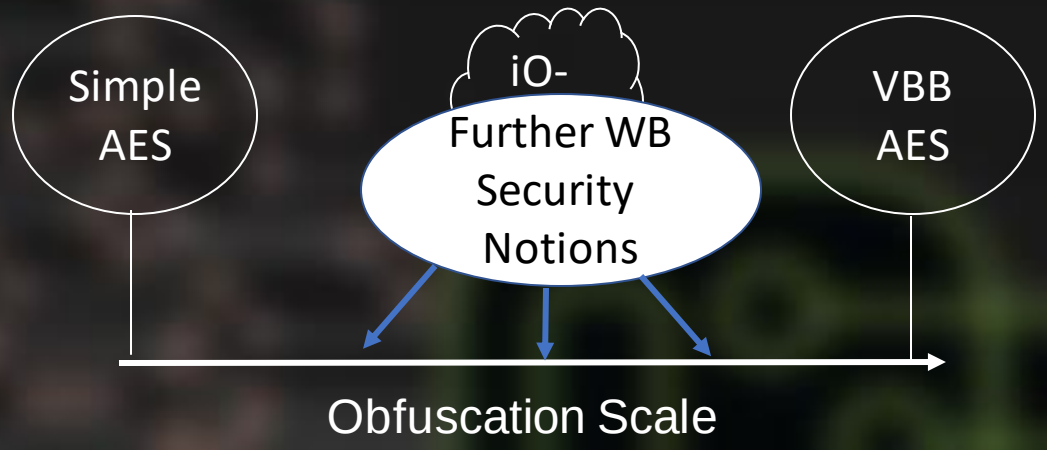
- Is iO-AES a right obfuscation notion?
  - Not clear to the community
  - iO security does not ensure that an iO-obfuscated encryption program resists key extraction

WBC Security Notions



- We need something
  - relaxed compared to VBB
  - meaningful compared to iO

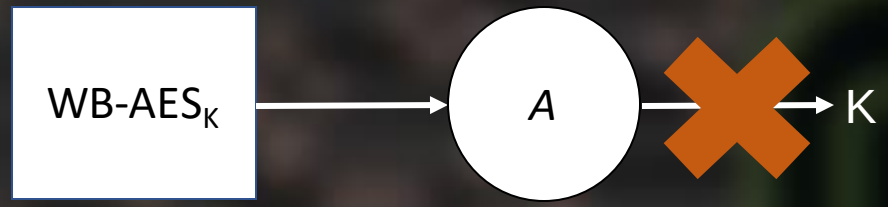
WBC Security Notions



- We need something
  - relaxed compared to VBB
  - meaningful compared to iO ⟹ further notions

WBC Security Notions

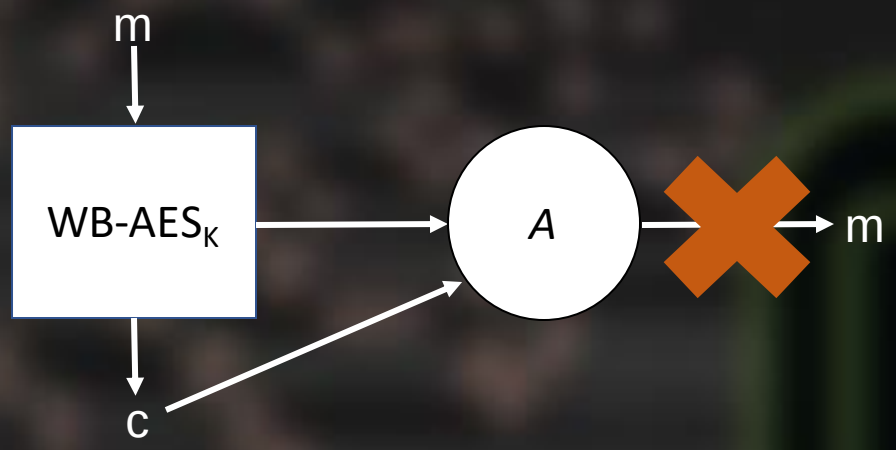
- Unbreakability: Keys can not be extracted



- Basic requirement but not sufficient depending on applications
- White-Box Security Notions for Symmetric Encryption Schemes (SAC 2013)
- Towards Security Notions for White-Box Cryptography (ISC 2009)

WBC Security Notions

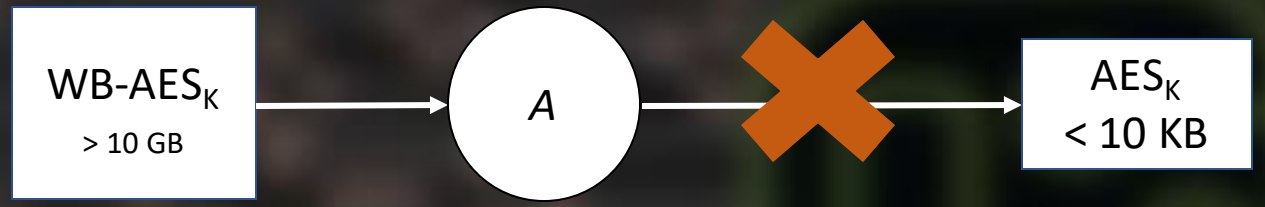
- One-Wayness: hardness of inversion



- Turns AES into a public-key cryptosystem
- Application: Public Key crypto with light-weight private operations

WBC Security Notions

- Incompressibility: hardness of compression

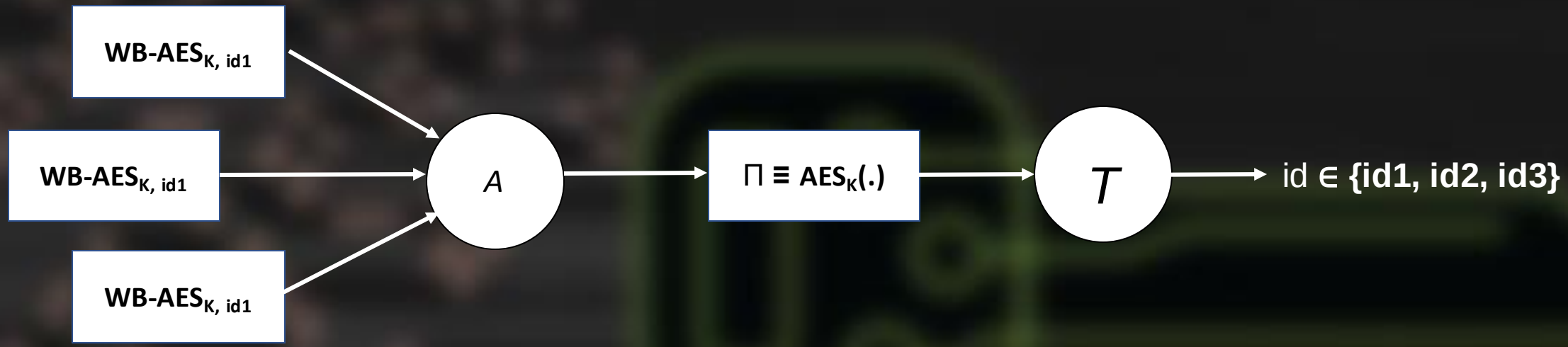


- Prevents Code-Lifting. Makes sharing the code at a large scale less convenient

- Some primitives based on this idea
  - Block-ciphers based on large tables (ASIACRYPT 2016)
  - Big-key cipher (CRYPTO 2016)

WBC Security Notions

- Traceability: Include a tracing mechanism



$\exists\ T$ such that $\forall\ A:$ **WB-AES$_{K, id1}$** $\mapsto$ **Π ≡ AES$_K$(.)** $\Rightarrow T\ (\Pi) = \mathrm{id}$

- Use Case: Pay-TV

Outline of the Talk

1 Intro

2 Obfuscation

3 White Box Crypto

4 White Box AES

White Box AES (Chow et al., SAC 2002)

My Assumption: AES is Known to Everyone

- First significant work on White-box AES
- Two steps:

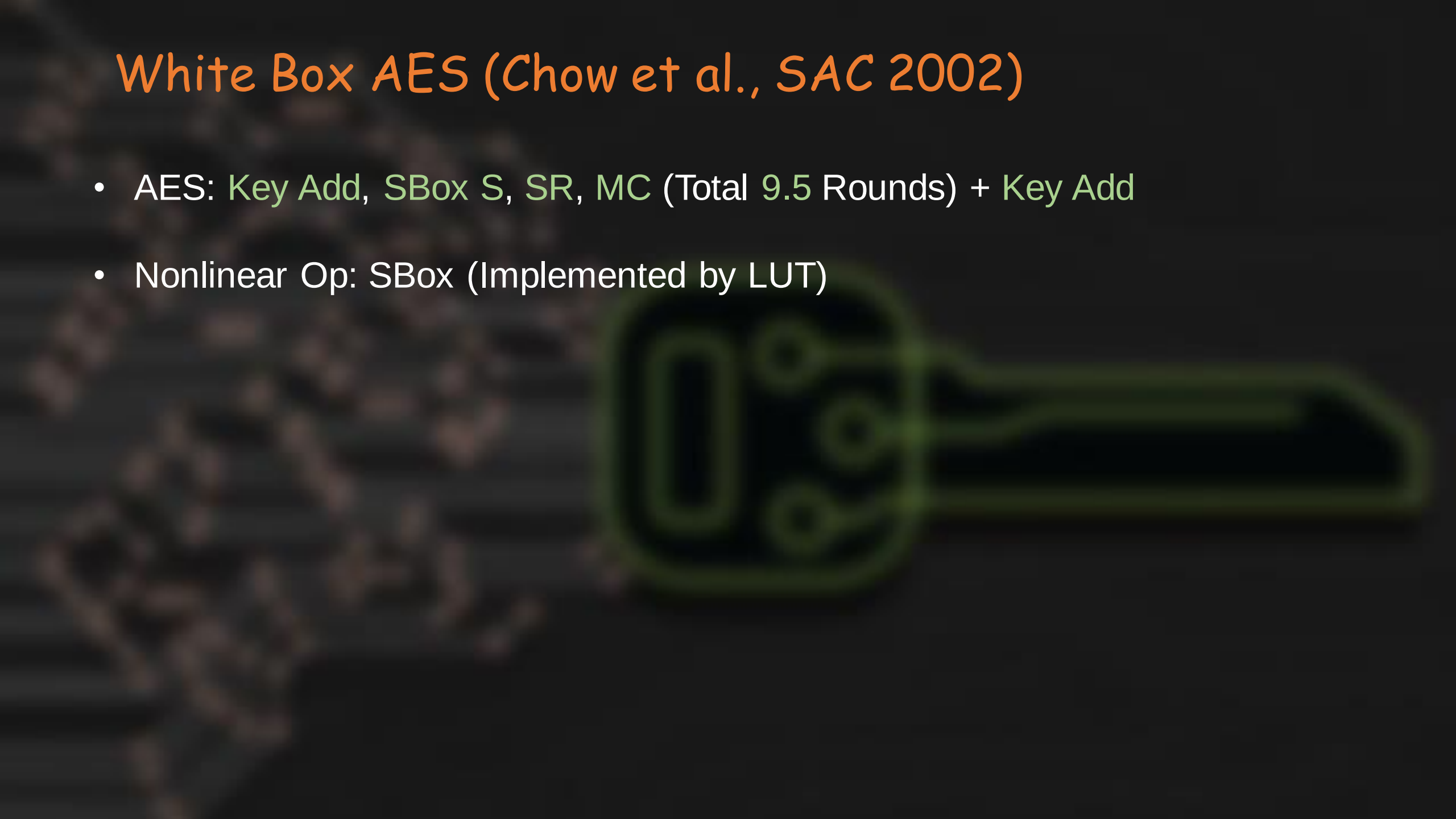
  - First step: represent AES as a network of look-up tables

  - Randomize the look-up tables (with random encodings)

  - Intuition: encoded tables bring no information on the key

White Box AES (Chow et al., SAC 2002)

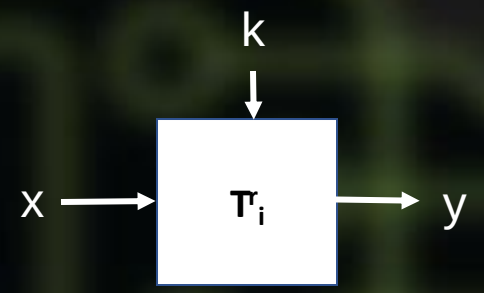
- AES: Key Add, SBox S, SR, MC (Total 9.5 Rounds) + Key Add
- Nonlinear Op: SBox (Implemented by LUT)



White Box AES (Chow et al., SAC 2002)

- AES: Key Add, SBox S, SR, MC (Total 9.5 Rounds) + Key Add
- Nonlinear Op: SBox (Implemented by LUT)
- WB-AES plays the Trick

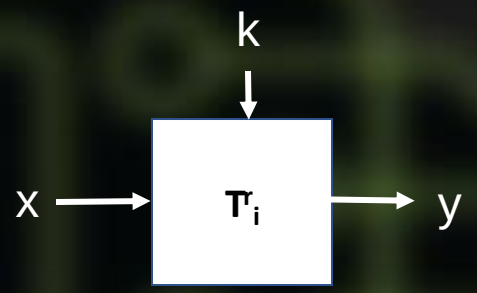
  - Use $TBox_k$ (instead of SBox)



White Box AES (Chow et al., SAC 2002)

- AES: Key Add, SBox S, SR, MC (Total 9.5 Rounds) + Key Add
- Nonlinear Op: SBox (Implemented by LUT)
- WB-AES plays the Trick

  - Use $TBox_k$ (instead of SBox)



  - Overall:

$$T^r_{i,k}(x) = S(x + k^{r-1}[i]), \quad i = 0, \ldots, 15, r = 1, \ldots, 9 \text{ (144 tables)}$$
$$T^{10}_{i,k}(x) = S(x + k^9[i] + k^{10}[i]), \quad i = 0, \ldots, 15 \text{ (16 tables)}$$

- Total 160 tables: 8 bits to 8 bits

White Box AES (Chow et al., SAC 2002)

- T_{yi} Tables for MixColumns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} = x0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} + x1 \begin{bmatrix} 03 \\ 01 \\ 01 \\ 02 \end{bmatrix} + x2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + x3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$

White Box AES (Chow et al., SAC 2002)

- T_{yi} Tables for MixColumns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} = x0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} + x1 \begin{bmatrix} 03 \\ 01 \\ 01 \\ 02 \end{bmatrix} + x2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + x3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$

- Table $T_{y0}(x) = x\,[02\ \ 01\ \ 01\ \ 03]^T$, similar for $T_{y1}(x)$, $T_{y2}(x)$, $T_{y3}(x)$
- One Column: $T_{y0}(x0) + T_{y1}(x1) + T_{y2}(x2) + T_{y3}(x3)$

White Box AES (Chow et al., SAC 2002)

- T_{yi} Tables for MixColumns

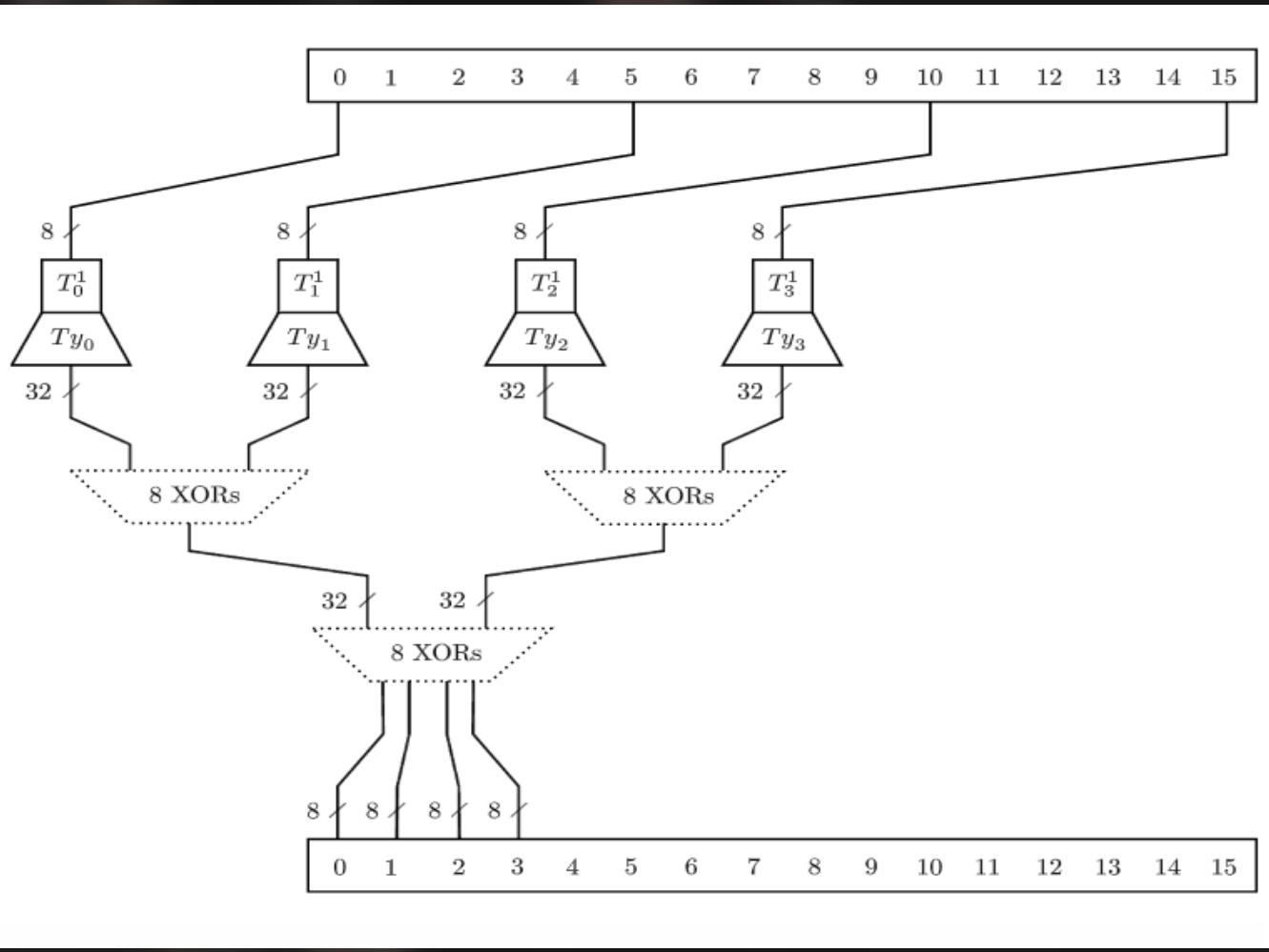
$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \end{bmatrix} = x0 \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} + x1 \begin{bmatrix} 03 \\ 01 \\ 01 \\ 02 \end{bmatrix} + x2 \begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} + x3 \begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix}$$

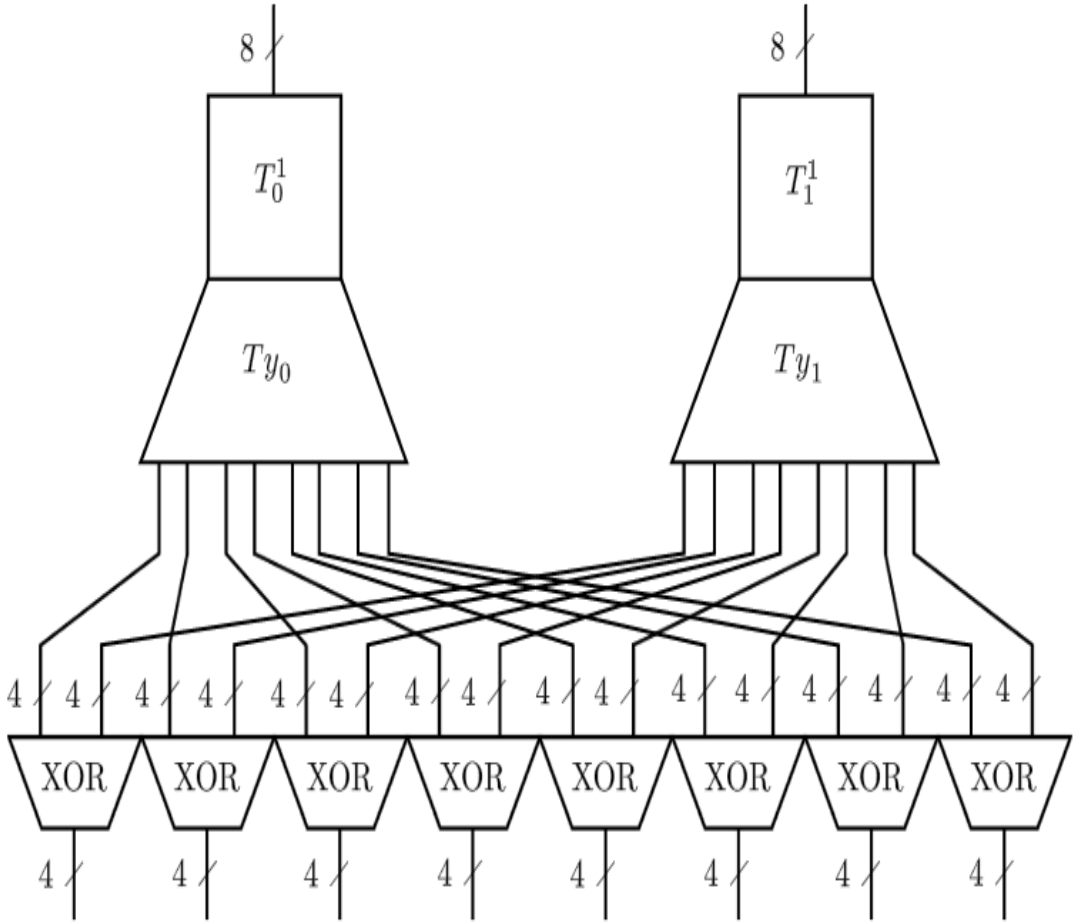
- Table $T_{y0}(x) = x\,[02\ \ 01\ \ 01\ \ 03]^T$, similar for $T_{y1}(x)$, $T_{y2}(x)$, $T_{y3}(x)$
- One Column: $T_{y0}(x0) + T_{y1}(x1) + T_{y2}(x2) + T_{y3}(x3)$
- Total 144 tables: 36 (9 rounds, 4 columns) tables for each $T_{yi}(x)$

White Box AES (Chow et al., SAC 2002)

- XOR Tables: XOR (x, y) = x + y
- Total 864 tables (8 bit to 4 bit tables, x and y are 4 bits)

White Box AES (Chow et al., SAC 2002)





White Box AES (Chow et al., SAC 2002)

- Randomize each table T by

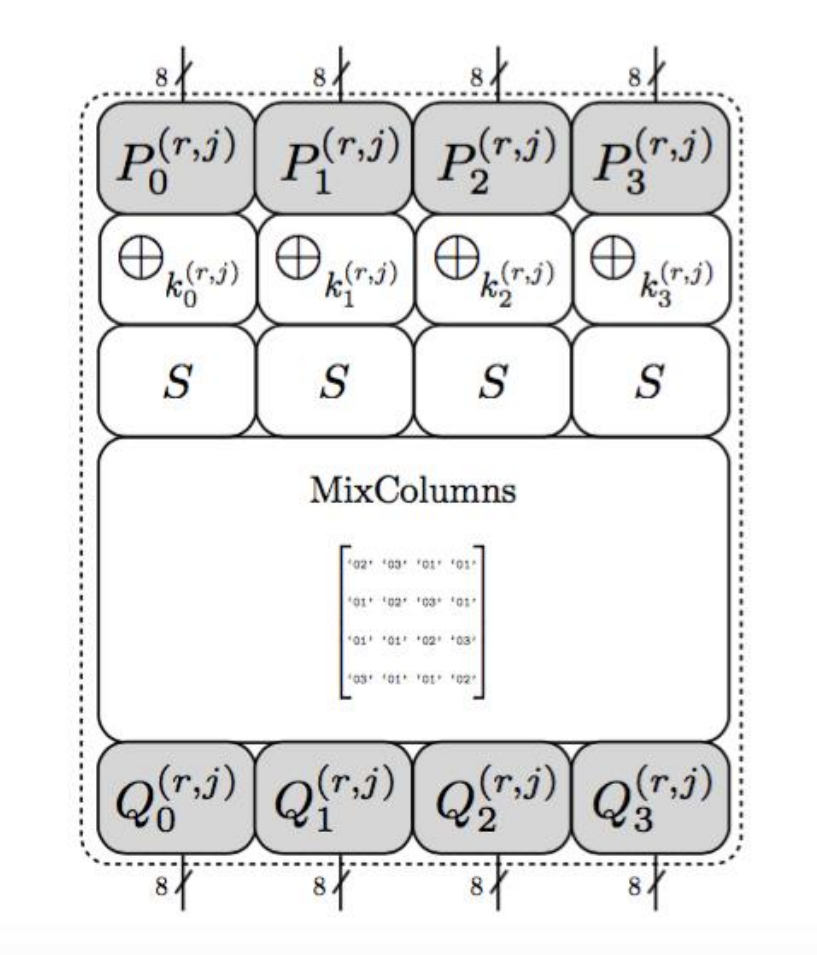
  $T' = g \circ T \circ f^{-1}$, where g and f^{-1} are random encodings

- For two connected tables T and S,

  $T' = g \circ T \circ f^{-1}$
  $S' = h \circ S \circ g^{-1}$
  $S' \circ T' = h \circ (S \circ T) \circ f^{-1}$

- Intuition: encoded tables bring no information on the key
  - True for a single table
  - $h \circ (Tn \circ Tn\text{-}1 \circ \ldots\ldots \circ T1) \circ f^{-1}$
  - Not true for a larger picture (Attacks exist)

Encoded sub-round (32 bits to 32 bits)



The four key bytes can be extracted from it (Lepoint et al., SAC 2013)

Attacks on Orginal WB-AES and Attempts

Many attacks on the original design:

- First break: BGE attack (Billet et al. SAC 2004)
- Generic attack on WB SPN ciphers (Michiels et al. SAC 2008)
- Improved BGE attack (Lepoint et al. SAC 2013)
- Collision attack (Lepoint et al. SAC 2013)
- Classical fault attacks (e.g. Piret-Quisquater, CHES 2003)
- Dierential Computational Analysis (Bos et al. CHES 2016)

Attempts

- Perturbed WB-AES using MV crypto (Bringer et al. ePrint 2006)

broken (De Mulder et al. INDOCRYPT 2010)

- WB-AES based on wide linear encodings (Xiao-Lai et al. CSA 2009)

broken (De Mulder et al. SAC 2012)

- WB-AES based on dual AES ciphers (Karroumi, ICISC 2010)

broken (Lepoint et al. SAC 2013)

Documents Needed for this Presentation

- A Tutorial on White-box AES, James A Muir, https://eprint.iacr.org/2013/104.pdf

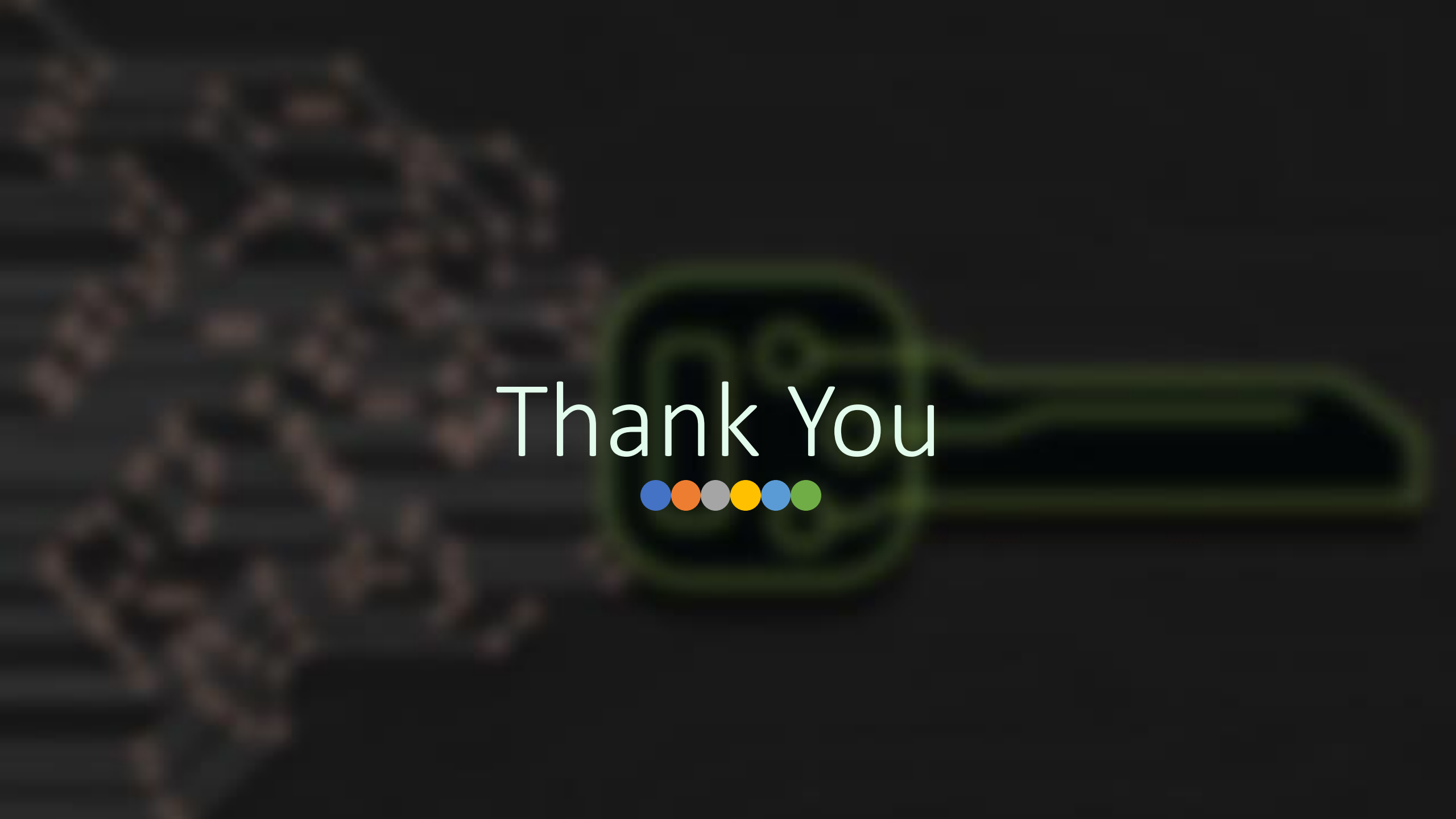
- White Box Crypto, Matthieu Rivain, https://www.matthieurivain.com/files/slides-cardis17.pdf

- Keynote: White-Box Cryptography, Matthieu Rivain, https://www.matthieurivain.com/files/slides-phisic16.pdf

- White-box Cryptography – New Challenges and Research Directions, https://www.ecrypt.eu.org/csa/documents/D1.3-WhiteboxCrypto-1.0.pdf

- On the (Im)possibility of Obfuscating Programs, Barak Et al., https://www.iacr.org/archive/crypto2001/21390001.pdf

- White-Box Cryptography and an AES Implementation, Chow et al., https://www.researchgate.net/publication/221274739_White-Box_Cryptography_and_an_AES_Implementation


Thank You