CRYPTOEXPERTS

WE INNOVATE TO SECURE YOUR BUSINESS

White-Box Security Notions for Symmetric Encryption Schemes

Cécile Delerablée[1] **Tancrède Lepoint**[1,2]
Pascal Paillier[1] **Matthieu Rivain**[1]

CryptoExperts[1], École Normale Supérieure[2]

SAC 2013

Outline

What is NOT white-box crypto?

General obfuscation

- from **any** program P, generate an obfuscated program $O(P)$
- hide **any** program property π in the code of $O(P)$
- meaning: the code of $O(P) \equiv$ a black-box oracle that runs P

How realistic is obfuscation?

- **very** strong requirements on the compiler O
- known impossibility results [BGI+01]

What is white-box crypto?

$\neq$ general program obfuscation!

White-box cryptography [CEJO+02]

- considers programs in a **restricted** class

$$\texttt{programs}(f) \quad \text{where} \quad f = \text{some keyed function}$$

- hides **some** program properties π in the code (but not all)
- code $\equiv$ a black-box oracle **only in some adversarial contexts**
- already provably secure constructions for some f
  (f = re-encryption [HRSV07,CCV12])
- no impossibility results so far for f = blockcipher
- but **no secure** construction for e.g. $f = AES_k(\cdot)$, $k \leftarrow \$$

Our approach

What do we really want from white-box crypto?

1. given $k \leftarrow \$$, generate (possibly randomly) $P = [AES_k(.)]$
2. it must be hard to recover k by playing around with P OLD
3. it also must be hard to decrypt under k OLD
4. we may want P to be **big** and **incompressible** NEW
5. we may want to distribute **traceable** NEW versions $P_1, \ldots, P_n$

This work

- we capture 1-5 into **concrete security games** OLD+NEW
- we build a toy blockcipher that provably satisfies 1-4 NEW
- we build a construction that provably achieves 5 NEW

Outline

White-box compilers

Let $\mathcal{E} = (K, E, D)$ be a symmetric encryption scheme.

Definition

A white-box compiler $\mathbf{C}_{\mathcal{E}}$ takes as input a key $k \in \mathrm{K}$ and some index $r \in \mathrm{R}$ and outputs a program $P = \mathbf{C}_{\mathcal{E}}(k, r) = [E_k^r]$.

Huge behavioral differences between

function $E(\cdot, \cdot)$	**oracle** $E(k, \cdot)$	**program** $[E_k^r]$
analytic description or algorithmic description	remote access, input/output only, might be stateful	word in a language, stateless since rebootable, copiable, transferable, observable, modifiable, system calls simulatable
(specification)	(smart card)	(executable software)

Attack models

Security notion = adversarial goal + attack model

What are the attack models against white-box programs?

Given the description of $\mathbf{C}_{\mathcal{E}}(\cdot,\cdot)$ and $P = [E_k^r]$ for unknown $k \in \mathrm{K}$

chosen-plaintext attack – CPA can encrypt any plaintext (unavoidable)

chosen-ciphertext attack – CCA can make decryption queries to an oracle $D(k,\cdot)$

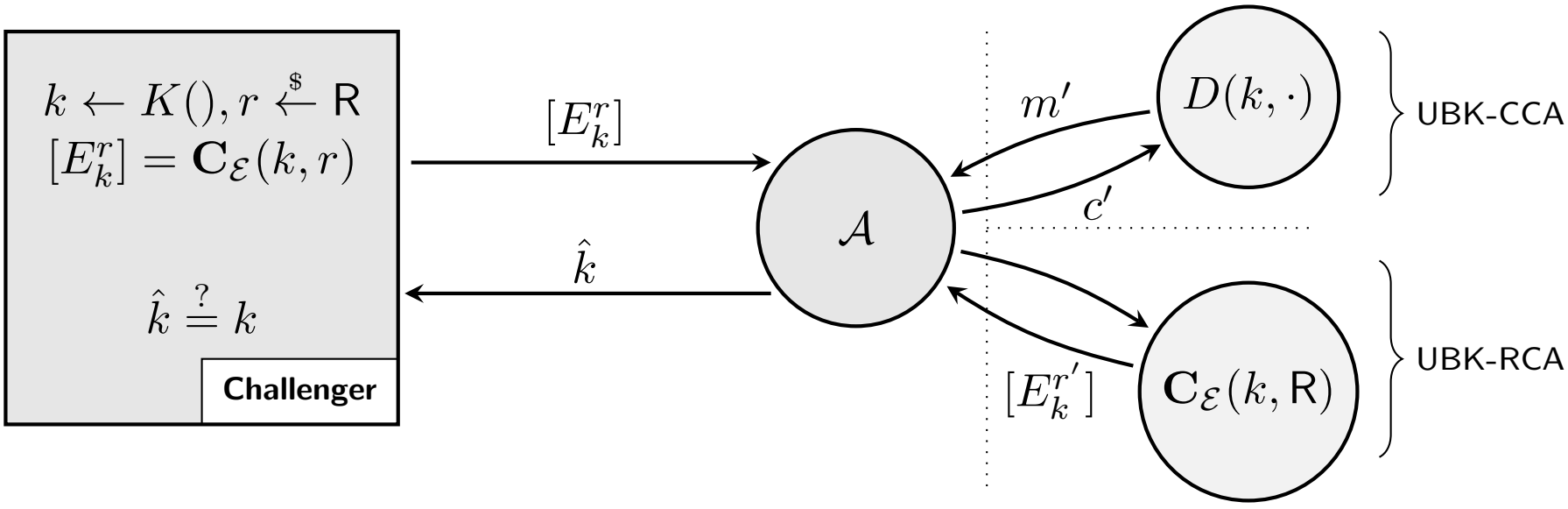
recompilation attack – RCA can make recompilation requests to get other programs $\mathbf{C}_{\mathcal{E}}(k,r')$ for unknown $r' \neq r$

combined attack – RCA + CCA most powerful (?)

RCA can be made stronger with known or chosen $r' \in \mathrm{R}$.

What about adversarial goals?

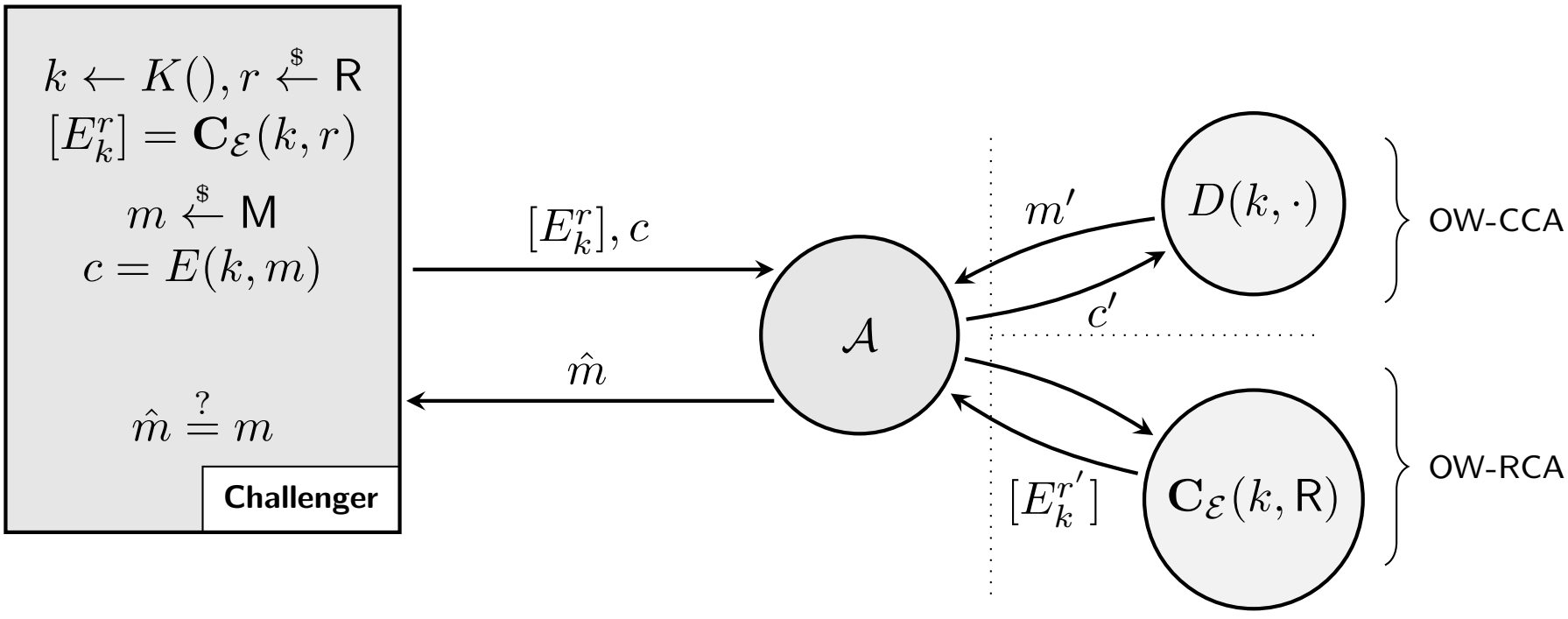
Unbreakability – UBK



There is no "semantic security" on k since verifying that $\hat{k} = k$ is easy.

So some information on k always leaks.

One-wayness – OW

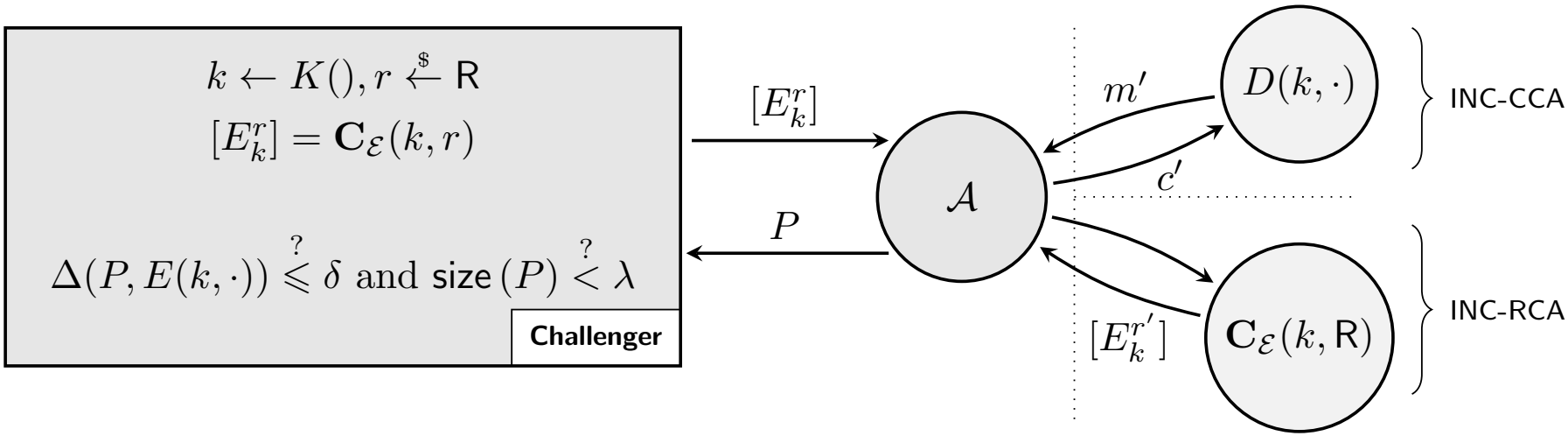


Again, no semantic security on m since verifying that $\hat{m} = m$ is easy.

Expected since $\mathcal{E}$ is a deterministic encryption scheme.

Incompressibility – INC

Given a large program, build an equivalent yet much smaller one



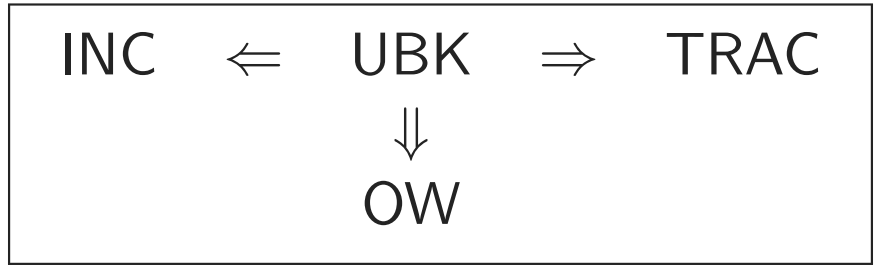
Traceability – TRAC

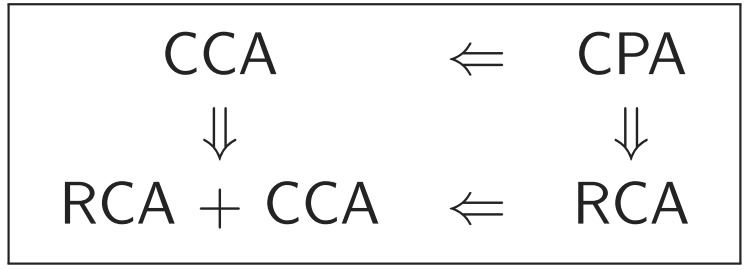
$\mathbf{C}_{\mathcal{E}}$ admits a **tracing scheme** if there exists an algorithm `trace` such that no adversary can win the "tracing game" TRAC:

- generate a key $k \xleftarrow{\$} \mathrm{K}$ and $P_1 = [E_k^{r_1}], \ldots, P_n = [E_k^{r_n}]$
- $\mathcal{A}$ chooses some $T \subseteq [1, n]$ and is provided with $\{P_i, i \in T\}$
- $\mathcal{A}$ returns some rogue program $Q \leftarrow \mathcal{A}(\{P_i, i \in T\})$
- trace a traitor $t \leftarrow \mathtt{trace}(Q, k, r_1, \ldots, r_n)$
- $\mathcal{A}$ wins if Q is functional enough and $t \notin T$

The big picture

$\alpha \Leftarrow \beta$: if β can be broken, α can be broken





The weakest security notion is UBK-CPA.
We don't even know how to achieve it with $\mathcal{E} = AES$...

Outline

Achieving incompressibility

A toy example...

$\mathcal{G}$ group of secret order w and e = exponent with large entropy

Hard problems on $\mathcal{G}$

Given $\mathtt{desc}(\mathcal{G})$ and e

- UBK$[\mathcal{G}]$ find the group order w (FACT)
- ORD$[\mathcal{G}]$ find the order of a group element ($\equiv$ FACT)
- ROOT$[\mathcal{G}, e]$ find the e-th root of a group element (RSA)
- GAP$[\mathcal{G}, e]$ find the group order w with the help of an e-th root extractor (FACT$^{\text{RSA}} \stackrel{\text{def}}{=}$ GAP-RSA)

Achieving incompressibility

Key generation: generate $k = (\mathtt{desc}(\mathcal{G}), e, w)$

Encryption: $E(k, m) = m^e$

Decryption: $D(k, c) = c^{1/e \bmod w}$

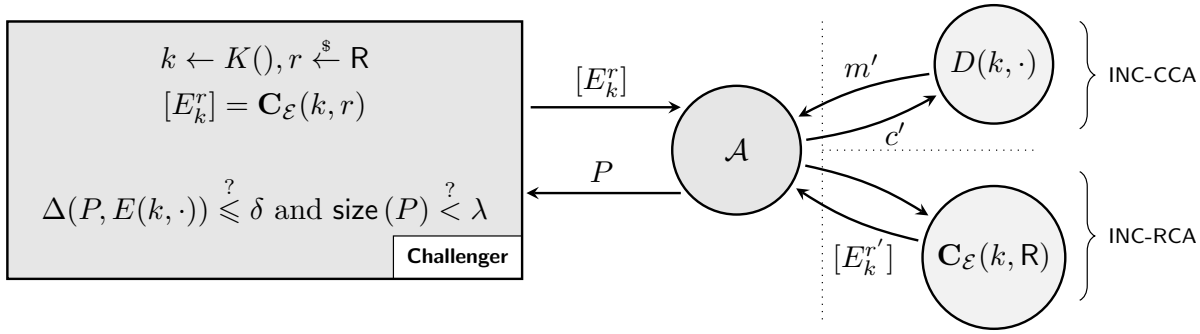
$$\boxed{\mathbf{C}_{\mathcal{E}}(k, r = \text{""}) \text{ just returns } [m \mapsto m^e]}$$

Then

$$\text{ORD}[\mathcal{G}] \Leftarrow \text{INC-CPA}$$

assuming that the compressed program is algebraic.

ORD[$\mathcal{G}$] $\Leftarrow$ INC-CPA



Here, $[E_k^r] = [m \mapsto m^e]$ and P is algebraic.

Using `extract`, we can find an execution of P where $P(m) = m^\alpha$ for a known α. Then

- either $\alpha \neq e$ then $e - \alpha \propto ord(m)$ and we break ORD[$\mathcal{G}$]
- or $\alpha = e$ then size $(P) \geqslant H(e)$ and P must be big

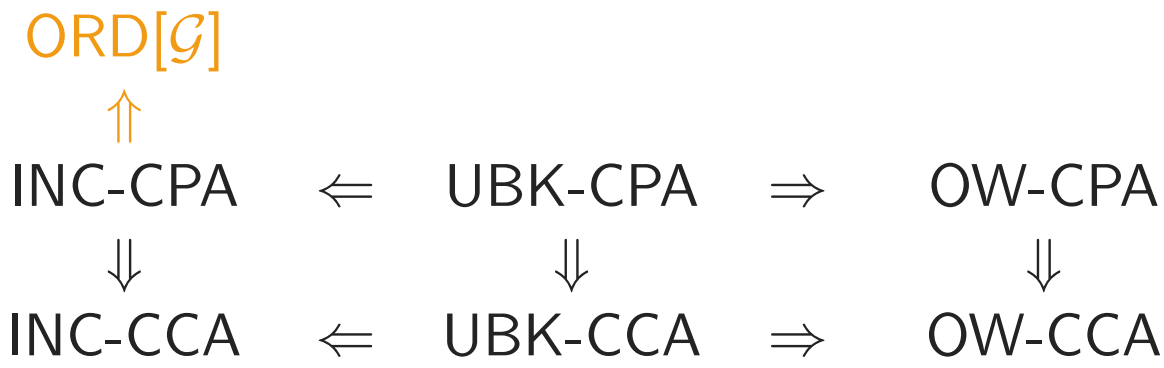
Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \Leftarrow & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \Leftarrow & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \end{array}$$

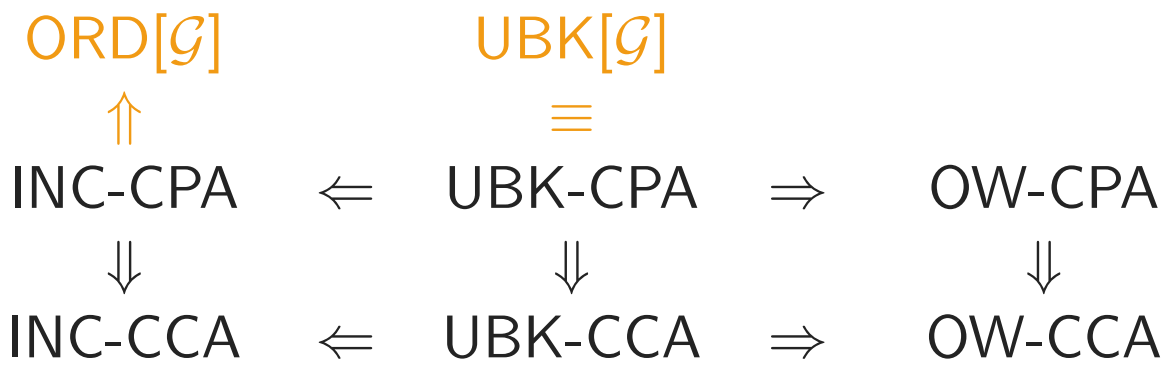
Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:



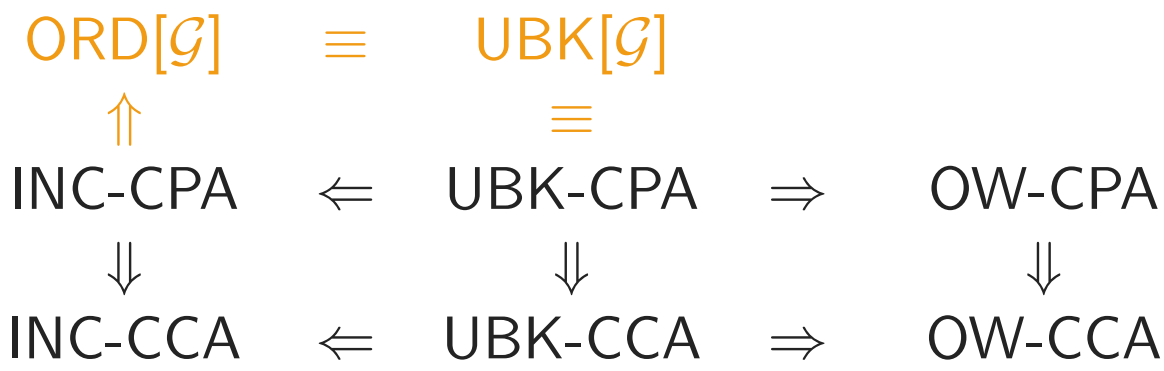
Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:



Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:



Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$
\begin{array}{ccccc}
\mathrm{ORD}[\mathcal{G}] & \equiv & \mathrm{UBK}[\mathcal{G}] & & \\
\equiv & & \equiv & & \\
\mathrm{INC\text{-}CPA} & \equiv & \mathrm{UBK\text{-}CPA} & \Rightarrow & \mathrm{OW\text{-}CPA} \\
\Downarrow & & \Downarrow & & \Downarrow \\
\mathrm{INC\text{-}CCA} & \Leftarrow & \mathrm{UBK\text{-}CCA} & \Rightarrow & \mathrm{OW\text{-}CCA}
\end{array}
$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \Leftarrow & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \end{array}$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$
\begin{array}{ccccc}
\text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\
\Downarrow & & \Downarrow & & \Downarrow \\
\text{INC-CCA} & \Leftarrow & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \\
 & & \equiv & & \\
 & & \text{GAP}[\mathcal{G}, e] & &
\end{array}
$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$
\begin{array}{ccccc}
\text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\
\Downarrow & & \Downarrow & & \Downarrow \\
\text{INC-CCA} & \Leftarrow & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \\
\Downarrow & & \equiv & & \\
\text{GAP}[\mathcal{G}, e] & & \text{GAP}[\mathcal{G}, e] & &
\end{array}
$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$
\begin{array}{ccccc}
\text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\
\Downarrow & & \Downarrow & & \Downarrow \\
\text{INC-CCA} & \equiv & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \\
\equiv & & \equiv & & \\
\text{GAP}[\mathcal{G}, e] & & \text{GAP}[\mathcal{G}, e] & &
\end{array}
$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \equiv & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \end{array}$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$
\begin{array}{ccccc}
 & & & & \mathsf{ROOT}[\mathcal{G}, e] \\
 & & & & \equiv \\
\mathsf{INC\text{-}CPA} & \equiv & \mathsf{UBK\text{-}CPA} & \Rightarrow & \mathsf{OW\text{-}CPA} \\
\Downarrow & & \Downarrow & & \Downarrow \\
\mathsf{INC\text{-}CCA} & \equiv & \mathsf{UBK\text{-}CCA} & \Rightarrow & \mathsf{OW\text{-}CCA}
\end{array}
$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \equiv & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \end{array}$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \equiv & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \\ & & & & \equiv \\ & & & & \text{easy} \end{array}$$

Achieving incompressibility

Security profile of $\mathbf{C}_{\mathcal{E}}$:

$$\begin{array}{ccccc} \text{INC-CPA} & \equiv & \text{UBK-CPA} & \Rightarrow & \text{OW-CPA} \\ \Downarrow & & \Downarrow & & \Downarrow \\ \text{INC-CCA} & \equiv & \text{UBK-CCA} & \Rightarrow & \text{OW-CCA} \end{array}$$

(under standard assumptions)

Outline

Traceable white-box programs

Assume we can hide "functional perturbations" in $[D_k^r]$

- a perturbation $c_i \mapsto m_i'$ means that $[D_k^r](c_i)$ returns m_i' instead of the correct plaintext $m_i = D(k, c_i)$
- the white-box compiler $\mathbf{C}_{\mathcal{E}}$ now takes a list of perturbations

$$(c_1 \mapsto m_1', c_2 \mapsto m_2', \ldots, c_u \mapsto m_u')$$

  as extra input
- assuming perturbations are "hidden", we can construct a log-efficient tracing scheme

Traceable white-box programs

Setup

User program	Specification	Perturbations
P_1	$[D(k,\cdot)]$	$c_1, c_2, \ldots, c_n$
P_2	$[D(k,\cdot)]$	$c_2, c_3, \ldots, c_n$
P_3	$[D(k,\cdot)]$	$c_3, c_4, \ldots, c_n$
$\vdots$	$\vdots$	$\vdots$
P_{n-1}	$[D(k,\cdot)]$	c_{n-1}, c_n
P_n	$[D(k,\cdot)]$	c_n

Note that

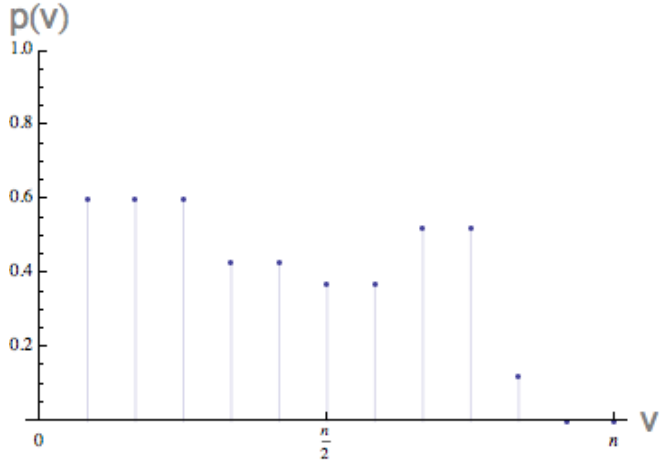
1. when $c \neq c_1, \ldots, c_n$, all programs decrypt c correctly
2. when $c = c_i$, programs $P_1, \ldots, P_i$ are incorrect on c but $P_{i+1}, \ldots, P_n$ are correct

Traceable white-box programs

We get a private-key linear broadcast encryption (PLBE) scheme

With

$$\begin{aligned} p(0) &= \Pr\left[Q(c) = D(k,c)\right] \quad \text{for } c \xleftarrow{\$} \mathrm{C} \\ p(v) &= \Pr\left[Q(c_v) = D(k,c_v)\right] \text{ for } v = 1,\ldots,n \end{aligned}$$



If there is a gap on the curve of $p(v)$ for some v then v is a traitor.

Traceable white-box programs

Tracing algorithm on rogue decryption program Q

Estimate $p(v)$ as $\hat{p}(v)$ and find a gap using dichotomy
$\Rightarrow$ takes $O(\log n)$ executions of Q

Requires 2 assumptions on "how well" perturbations are hidden by the white-box compiler.

See details in the paper.

Outline

Conclusion

New achievements

- framework of proper security notions for white-box compilers
- unbreakability + one-wayness + incompressibility is achievable
- traceability of programs is achievable under assumptions

A lot of issues remain

- are there any other security notions of interest?
  unforgeability? non-malleability? public verifiability?
- can we achieve any of these notions with a true blockcipher?
- . . . even just UBK-CPA with $f = AES$?
- can we extend traceability for $f =$ any keyed function?