

How to Trade Leakage for Tamper-Resilience

Daniele Venturi

Joint work with:

Sebastian Faust (Katholieke Universiteit Leuven)

Krzysztof Pietrzak (CWI Amsterdam)



SAPIENZA University of Rome

ICALP 2011 – Zurich, July 6 2011

Cryptography today: provable security



Cryptography today: provable security

1 Define model & security notion



Cryptography today: provable security

1 Define model & security notion

- This is done through a **security game** involving some



Cryptography today: provable security

1 Define model & security notion

- This is done through a **security game** involving some



2 Build cryptoscheme



Cryptography today: provable security

1 Define model & security notion

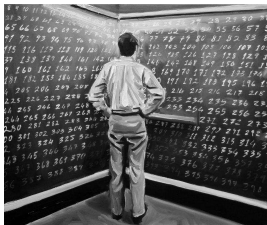
- This is done through a **security game** involving some



2 Build cryptoscheme



- ## 3 Formally prove security: Show that no (**efficient**) adversary can win the security game



Cryptography today: provable security

1 Define model & security notion

- This is done through a **security game** involving some

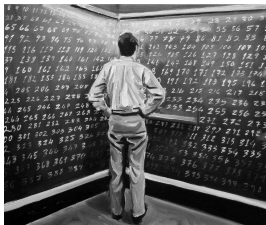


2 Build cryptoscheme



3 Formally prove security: Show that no (**efficient**) adversary can win the security game

- Often a **too strong** statement, as it e.g. implies $P \neq NP$ ☹



Cryptography today: provable security

1 Define model & security notion

- This is done through a **security game** involving some

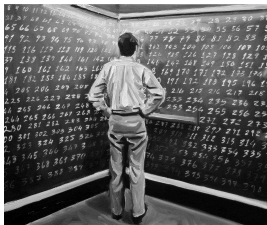


2 Build cryptoscheme



3 Formally prove security: Show that no (**efficient**) adversary can win the security game

- Often a **too strong** statement, as it e.g. implies $P \neq NP$ ☹️
- We can prove **conditional** result ☺️



Time to relax?

Security proof implies:



Time to relax?

Security proof implies:

- Security against **all known** and **future** attacks



Time to relax?



Security proof implies:

- Security against **all known** and **future** attacks
- Can we go home and relax?



Time to relax?



Security proof implies:

- Security against **all known** and **future** attacks
- Can we go home and relax?

- Provably secure systems **get broken in practice!**



Time to relax?



Security proof implies:

- Security against **all known** and **future** attacks
- Can we go home and relax?

- Provably secure systems **get broken in practice!**
- So what's wrong? Error in proof? Wrong assumption?



Time to relax?



Security proof implies:

- Security against **all known** and **future** attacks
- Can we go home and relax?

- Provably secure systems **get broken in practice!**
- So what's wrong? Error in proof? Wrong assumption?



A beautiful theory

public parameters



Adversary

secret state



Cryptosystem

- Security proofs usually rely on the **black-box model**



A beautiful theory

public parameters




Adversary

secret state



Cryptosystem

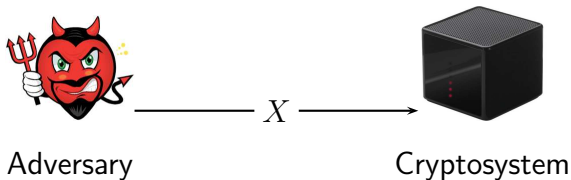
- Security proofs usually rely on the **black-box model**
-  has only **black-box** access to the cryptosystem




A beautiful theory

public parameters

secret state



- Security proofs usually rely on the **black-box model**
-  has only **black-box** access to the cryptosystem
 - he can specify an input X



A beautiful theory

public parameters

secret state




Adversary



Cryptosystem

Y

- Security proofs usually rely on the **black-box model**
-  has only **black-box** access to the cryptosystem
 - he can specify an input X
 - and gets the corresponding output Y



A beautiful theory

public parameters

secret state




Adversary

Y



Cryptosystem

- Security proofs usually rely on the **black-box model**
-  has only **black-box** access to the cryptosystem
 - he can specify an input X
 - and gets the corresponding output Y
 - the computations within the box stay secret



The cruel reality!

public parameters



secret state



The cruel reality!

public parameters



secret state



- In the real world the black box is actually a **physical device**



The cruel reality!


public parameters

secret state



X



- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time,



The cruel reality!


public parameters

secret state



X



- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound,




The cruel reality!

public parameters

secret state



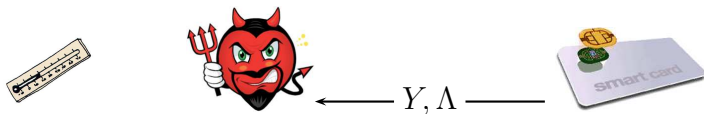
- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound, heat while the crypto-device is working




The cruel reality!

public parameters

secret state



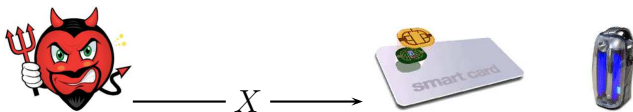
- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound, heat while the crypto-device is working
 - This results in a leakage Λ about the secret state. Even partial leakage suffices to break the cryptosystem [Kocher96]





The cruel reality!

public parameters

secret state



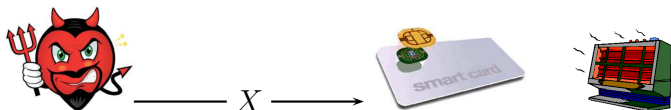
- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound, heat while the crypto-device is working
- **Active**  can apply **tampering attacks**: e.g. expose it to UV radiation,





The cruel reality!

public parameters

secret state



- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound, heat while the crypto-device is working
- **Active**  can apply **tampering attacks**: e.g. expose it to UV radiation, heating up the device



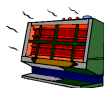
The cruel reality!



public parameters

secret state



← Y' →



- In the real world the black box is actually a **physical device**
- **Passive**  can apply **side-channel attacks**: e.g. measuring time, sound, heat while the crypto-device is working
- **Active**  can apply **tampering attacks**: e.g. expose it to UV radiation, heating up the device
 - The modified output can completely expose the secrets stored in the device [BDL00]

A general question



A general question

Question:

Consider **any** Boolean circuit C .



A general question

Question:

Consider **any** Boolean circuit C .

- C is a directed acyclic graph:
vertices \Leftrightarrow gates, edges \Leftrightarrow wires

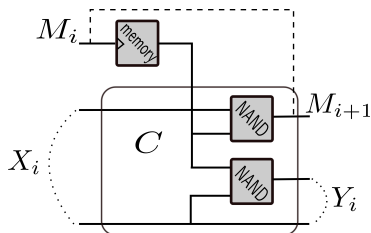


A general question

Question:

Consider **any** Boolean circuit C .

- C is a directed acyclic graph:
vertices \Leftrightarrow gates, edges \Leftrightarrow wires
- C can be **stateful**: input X_i and memory M_i are used to produce output Y_i and **new** state M_{i+1}

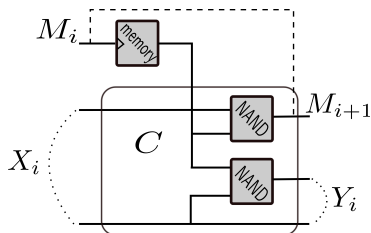


A general question

Question:

Consider **any** Boolean circuit C .

- C is a directed acyclic graph:
vertices \Leftrightarrow gates, edges \Leftrightarrow wires
- C can be **stateful**: input X_i and memory M_i are used to produce output Y_i and **new** state M_{i+1}
- C can be **randomized**

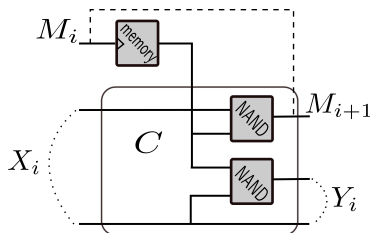


A general question

Question:

Consider **any** Boolean circuit C . Is it possible to formally prove that C is secure against an (**as large as possible**) class of fault attacks?

- C is a directed acyclic graph:
vertices \Leftrightarrow gates, edges \Leftrightarrow wires
- C can be **stateful**: input X_i and memory M_i are used to produce output Y_i and **new** state M_{i+1}
- C can be **randomized**



Compilers

A possible solution using the notion of **circuit compiler**:



Compilers

A possible solution using the notion of **circuit compiler**:

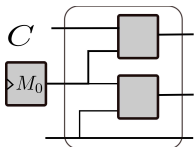
- Transform C in another circuit \hat{C} , in such a way that tampering in \hat{C} is detected with high probability



Compilers

A possible solution using the notion of **circuit compiler**:

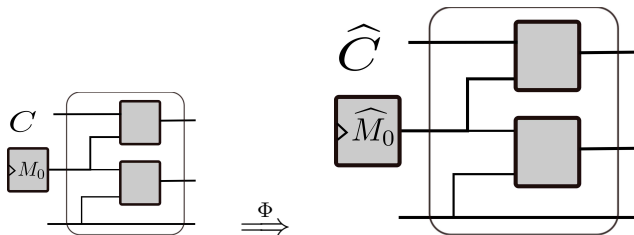
- Transform C in another circuit \hat{C} , in such a way that tampering in \hat{C} is detected with high probability



Compilers

A possible solution using the notion of **circuit compiler**:

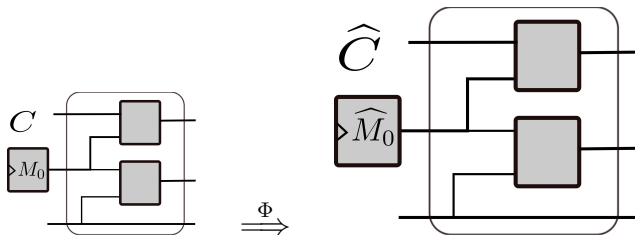
- Transform C in another circuit \hat{C} , in such a way that tampering in \hat{C} is detected with high probability



Compilers

A possible solution using the notion of **circuit compiler**:

- Transform C in another circuit \hat{C} , in such a way that tampering in \hat{C} is detected with high probability




- Φ is **functionality preserving**: C with initial state M_0 and \hat{C} with initial state \hat{M}_0 result in an **identical** output distribution

The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds




The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**





The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between






The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),







The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),  (i.e. **reset** a wire to 0)







The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),  (i.e. **reset** a wire to 0) and  (i.e. **flip** the value of a wire)







The “real” world

- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),  (i.e. **reset** a wire to 0) and  (i.e. **flip** the value of a wire)
- **Noisy Tampering**: each attack fails **independently** with some probability $0 < \delta \leq 1$








The “real” world

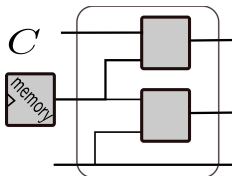
- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),  (i.e. **reset** a wire to 0) and  (i.e. **flip** the value of a wire)
- **Noisy Tampering**: each attack fails **independently** with some probability $0 < \delta \leq 1$
 - Faults can be either **permanent** or **transient**

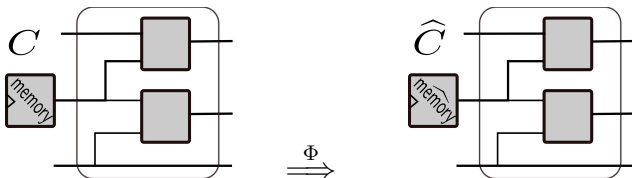


The “real” world

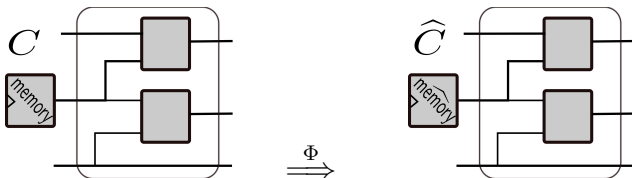
- Consider a **computationally unbounded** (∞, δ) -adversary tampering **adaptively** with \hat{C} for many rounds
- In each round  can attack **an unbounded number of wires**
 - For every wire he can choose between  (i.e. **set** a wire to 1),  (i.e. **reset** a wire to 0) and  (i.e. **flip** the value of a wire)
- **Noisy Tampering**: each attack fails **independently** with some probability $0 < \delta \leq 1$
 - Faults can be either **permanent** or **transient**
- Finally  gets the output of \hat{C} when tampering is applied to the computation

$(t, 0)$ -tamper resilience of [IPSW06]

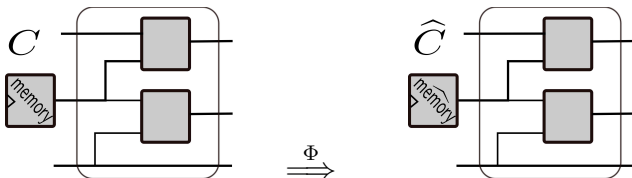
$(t, 0)$ -tamper resilience of [IPSW06]

$(t, 0)$ -tamper resilience of [IPSW06]

$(t, 0)$ -tamper resilience of [IPSW06]



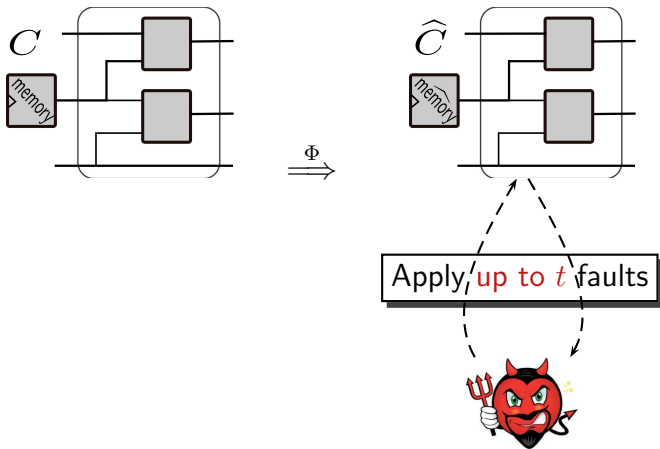
$(t, 0)$ -tamper resilience of [IPSW06]



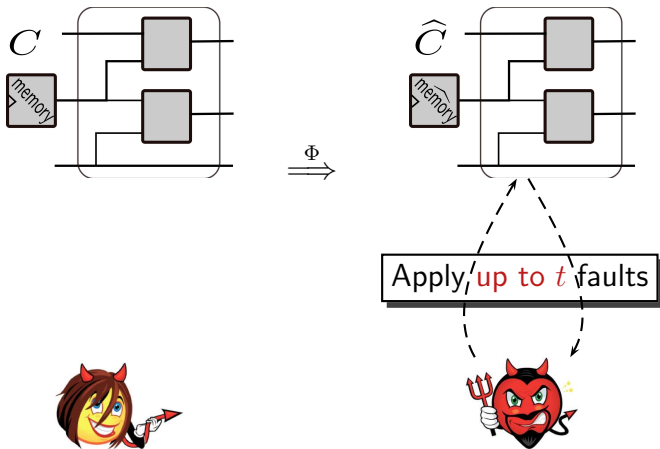
Apply up to t faults



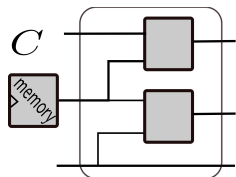
$(t, 0)$ -tamper resilience of [IPSW06]



$(t, 0)$ -tamper resilience of [IPSW06]



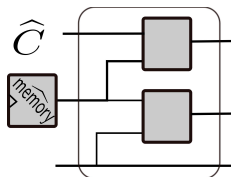
$(t, 0)$ -tamper resilience of [IPSW06]



Black box access



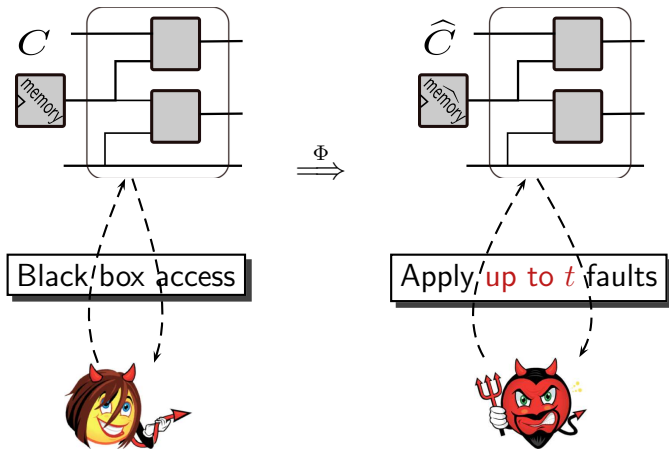
Φ



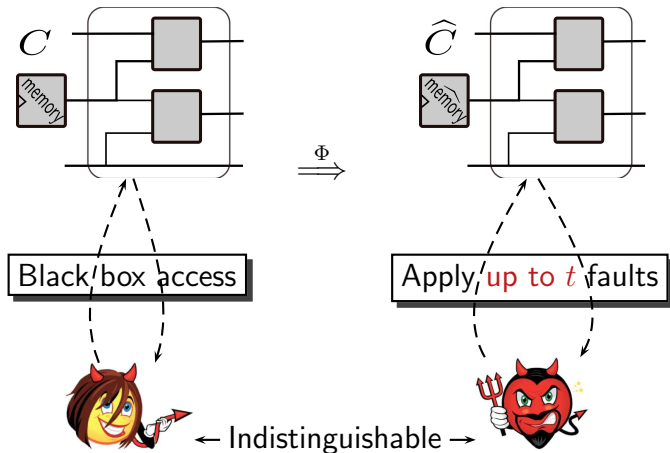
Apply up to t faults



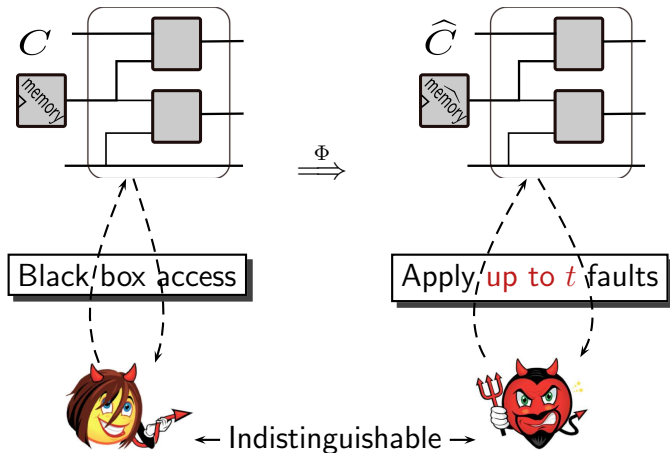
$(t, 0)$ -tamper resilience of [IPSW06]



$(t, 0)$ -tamper resilience of [IPSW06]



$(t, 0)$ -tamper resilience of [IPSW06]



Note: faults are error-free, i.e. $\delta = 0$

Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient



Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient
- Proof based on the following assumption



Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient
- Proof based on the following assumption

Axiom

There exist **small**, **stateless** and **computation-independent** tamper-proof “gadgets” computing with simple encodings



Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient
- Proof based on the following assumption

Axiom

There exist **small**, **stateless** and **computation-independent** tamper-proof “gadgets” computing with simple encodings

- **Inefficient** compiler. To achieve indistinguishability of 2^{-k}



Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient
- Proof based on the following assumption

Axiom

There exist **small**, **stateless** and **computation-independent** tamper-proof “gadgets” computing with simple encodings

- **Inefficient** compiler. To achieve indistinguishability of 2^{-k}
 - Blow-up is $O(k^3t)$



Result of [IPSW06]

- Theorem: For integer t and security parameter k , there exists a compiler that is $(t, 0)$ -tamper resilient
- Proof based on the following assumption

Axiom

There exist **small**, **stateless** and **computation-independent** tamper-proof “gadgets” computing with simple encodings

- **Inefficient** compiler. To achieve indistinguishability of 2^{-k}
 - Blow-up is $O(k^3t)$
 - Requires $O(k^2)$ bits of fresh randomness **per invocation**



Rest of this talk

- 1 Our paradigm: trading leakage for efficiency



Rest of this talk

- 1 Our paradigm: trading leakage for efficiency
- 2 Description of our compiler



Rest of this talk

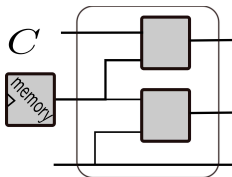
- 1 Our paradigm: trading leakage for efficiency
- 2 Description of our compiler
- 3 Proof sketch

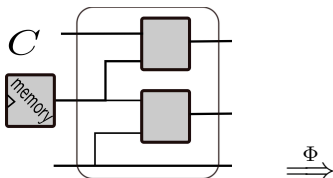


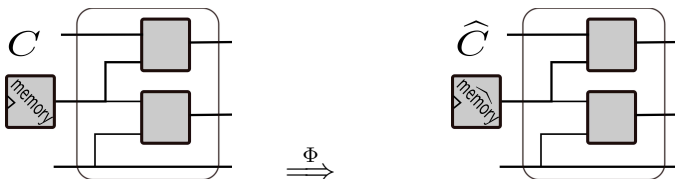
Rest of this talk

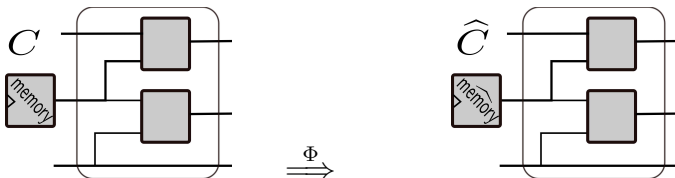
- 1 Our paradigm: trading leakage for efficiency
- 2 Description of our compiler
- 3 Proof sketch
- 4 Conclusions and perspective

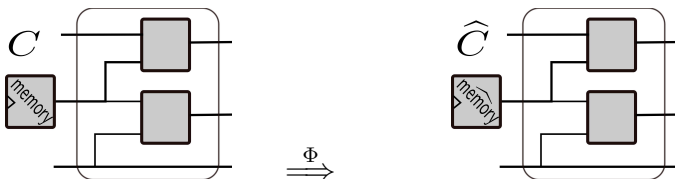


$(\infty, \delta, \lambda)$ -tamper resilience

$(\infty, \delta, \lambda)$ -tamper resilience

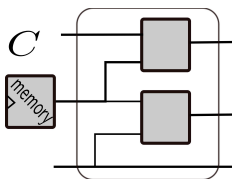
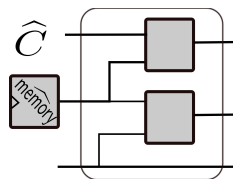
$(\infty, \delta, \lambda)$ -tamper resilience


$(\infty, \delta, \lambda)$ -tamper resilience


$(\infty, \delta, \lambda)$ -tamper resilience


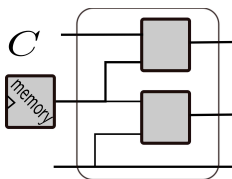
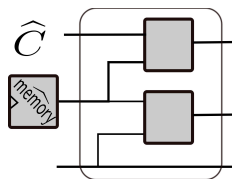
Apply **unbounded** # faults



$(\infty, \delta, \lambda)$ -tamper resilience

 \Rightarrow


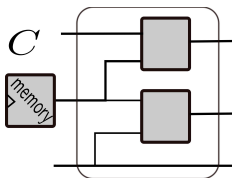
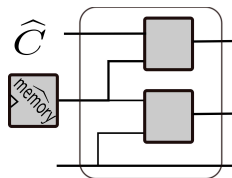
Apply **unbounded** # faults



$(\infty, \delta, \lambda)$ -tamper resilience

 Φ


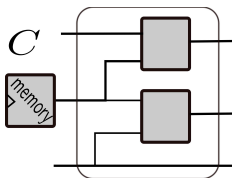
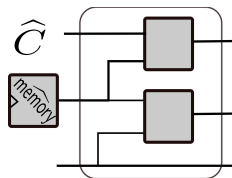
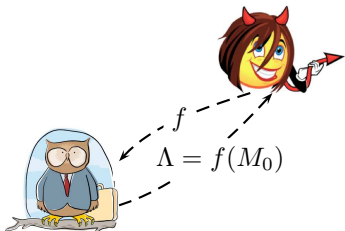
Apply **unbounded #** faults

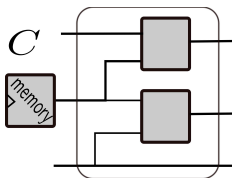
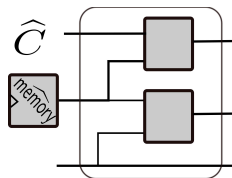


$(\infty, \delta, \lambda)$ -tamper resilience

 \Rightarrow


Apply **unbounded #** faults



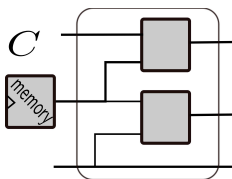
$(\infty, \delta, \lambda)$ -tamper resilience

 Φ

 Apply **unbounded #** faults


$(\infty, \delta, \lambda)$ -tamper resilience

 Φ

 Apply **unbounded #** faults

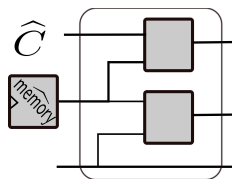

$$|\Lambda| = \lambda$$



$$\Lambda = f(M_0)$$

$(\infty, \delta, \lambda)$ -tamper resilience


Black box access

 Φ


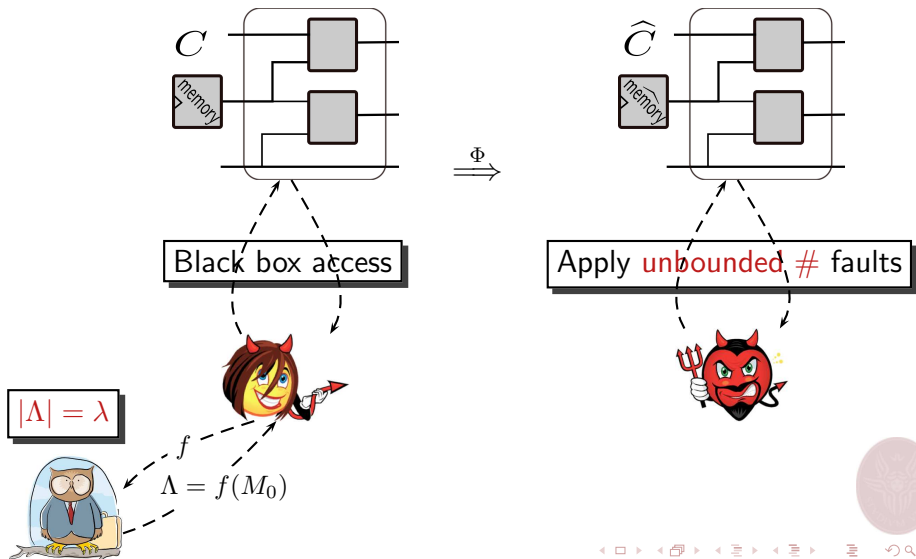
Apply **unbounded #** faults

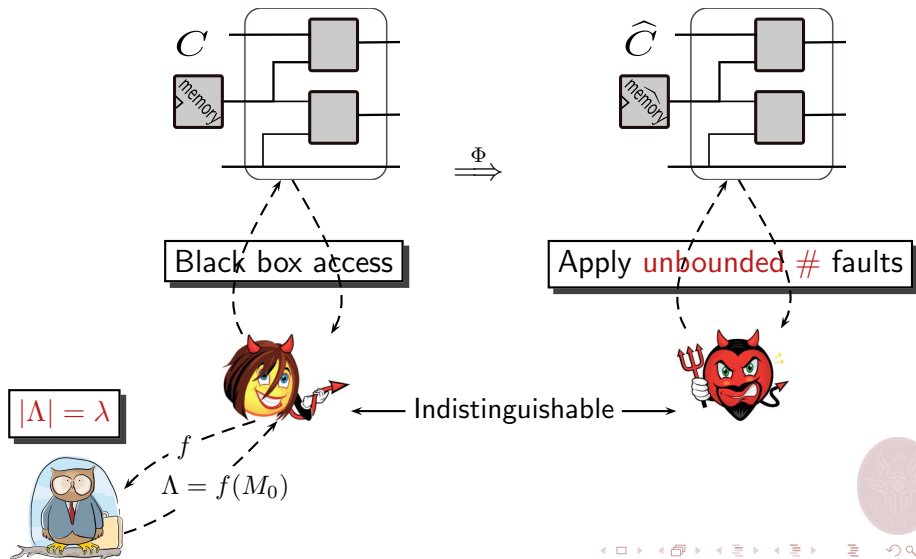
$$|\Lambda| = \lambda$$



$$\Lambda = f(M_0)$$



$(\infty, \delta, \lambda)$ -tamper resilience


$(\infty, \delta, \lambda)$ -tamper resilience


Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)
 - Blow-up is only $O(k)$ ☺



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)
 - Blow-up is only $O(k)$ ☺
 - **No randomness** needed at run-time ☺



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)
 - Blow-up is only $O(k)$ ☺
 - **No randomness** needed at run-time ☺
- Corollary: Any **scheme** tolerating a **logarithmic** amount of leakage on the secret key can be implemented in a tamper-resilient way



Our result

- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)
 - Blow-up is only $O(k)$ ☺
 - **No randomness** needed at run-time ☺
- Corollary: **Any scheme** tolerating a **logarithmic** amount of leakage on the secret key can be implemented in a tamper-resilient way
 - **Any** Sig and PKE (security loss exponential in leakage)

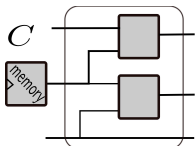


Our result

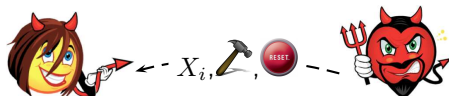
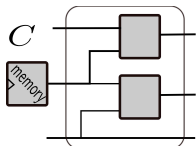
- Theorem: Let $\delta < 1/2$ and k be a security parameter. There exists a compiler that is $(\infty, \delta, O(\log|M_0|))$ -tamper resilient
- Comparison with [IPSW06]:
 - We rely on the **same axiom** and require similar tamper-proof components
 - $t = \infty$ but $\delta > 0$ (the two models are incomparable)
 - Blow-up is only $O(k)$ ☺
 - **No randomness** needed at run-time ☺
- Corollary: **Any scheme** tolerating a **logarithmic** amount of leakage on the secret key can be implemented in a tamper-resilient way
 - **Any** Sig and PKE (security loss exponential in leakage)
 - Positive results from **leakage-resilient cryptography**



What do we want from \hat{C} ?



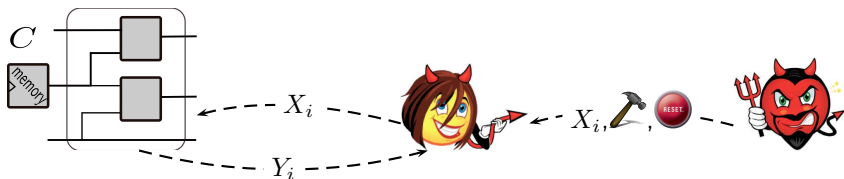
What do we want from \hat{C} ?



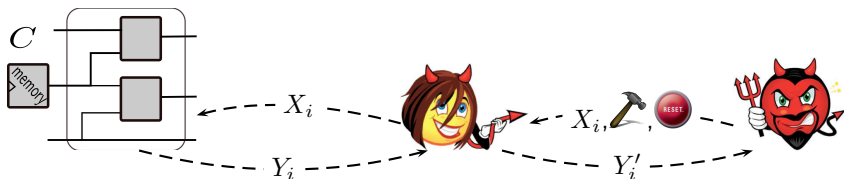
What do we want from \hat{C} ?



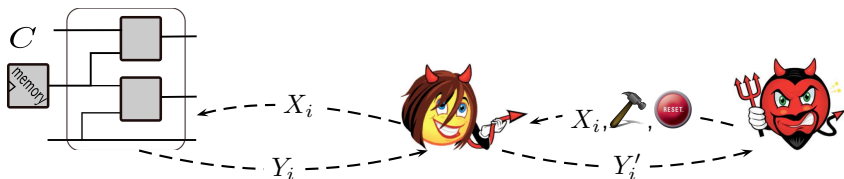
What do we want from \hat{C} ?



What do we want from \hat{C} ?



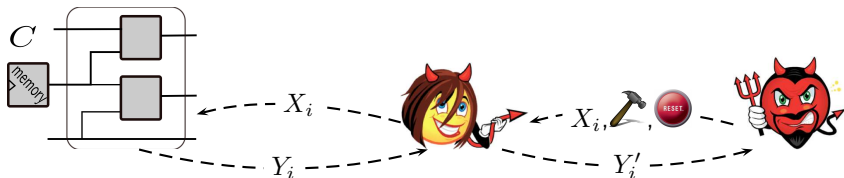
What do we want from \hat{C} ?



- Simulation is hard because



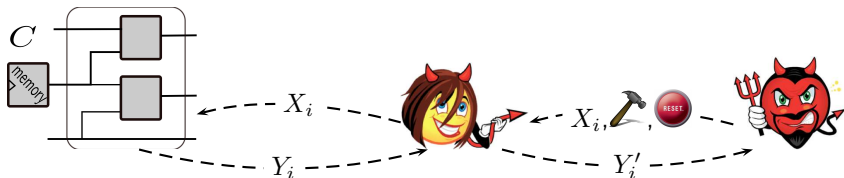
What do we want from \hat{C} ?



- Simulation is hard because
 - Y_i can't be directly forwarded



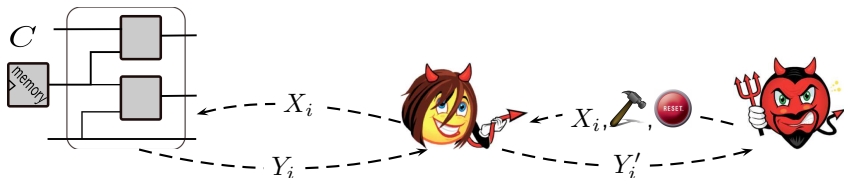
What do we want from \hat{C} ?



- Simulation is hard because
 - Y_i can't be directly forwarded
 - M_i is unknown



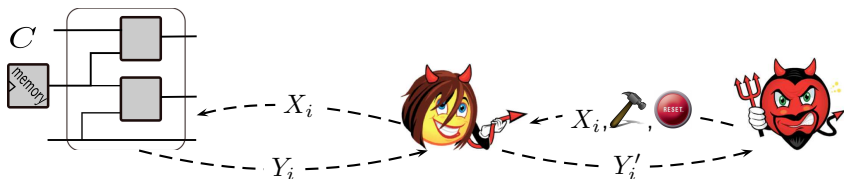
What do we want from \hat{C} ?



- Simulation is hard because
 - Y_i can't be directly forwarded
 - M_i is unknown
- Idea: Guarantee that \hat{C} outputs



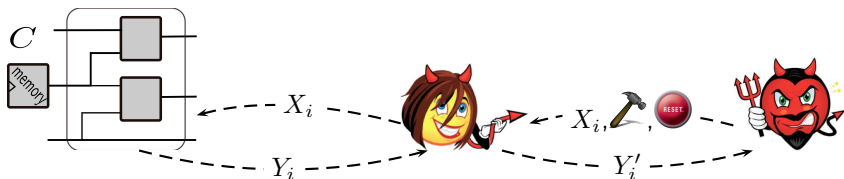
What do we want from \hat{C} ?



- Simulation is hard because
 - Y_i can't be directly forwarded
 - M_i is unknown
- Idea: Guarantee that \hat{C} outputs
 - Y_i when no tampering happens (easy to simulate)



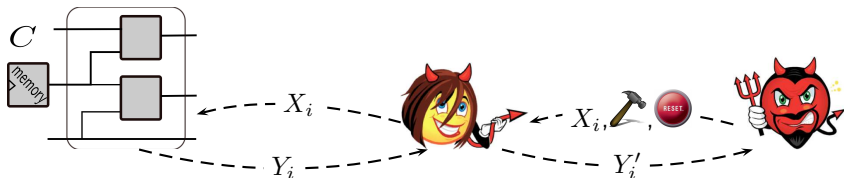
What do we want from \hat{C} ?



- Simulation is hard because
 - Y_i can't be directly forwarded
 - M_i is unknown
- Idea: Guarantee that \hat{C} outputs
 - Y_i when no tampering happens (easy to simulate)
 - Constant 0 if tampering occurs (we can reply with 0)

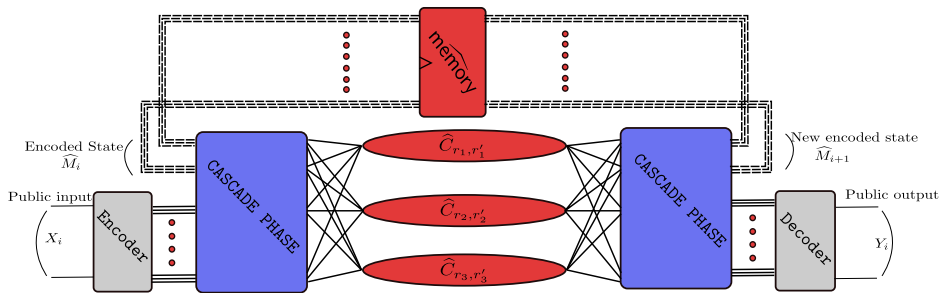


What do we want from \hat{C} ?

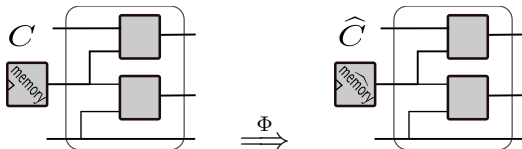


- Simulation is hard because
 - Y_i can't be directly forwarded
 - M_i is unknown
- Idea: Guarantee that \hat{C} outputs
 - Y_i when no tampering happens (easy to simulate)
 - Constant 0 if tampering occurs (we can reply with 0)
- Avoid: Tampering successfully without being noticed

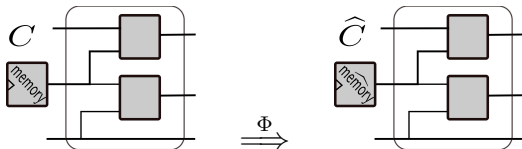


Big picture of \widehat{C} ($k = 3$)

The core (red part)



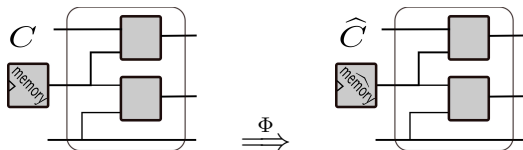
The core (red part)



- The core of \hat{C} consists of k sub-circuits (same topology as C)



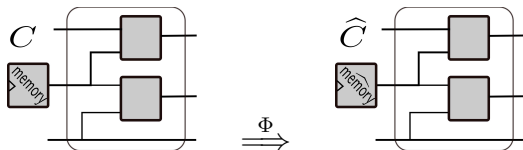
The core (red part)



- The core of \hat{C} consists of k sub-circuits (**same topology** as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$



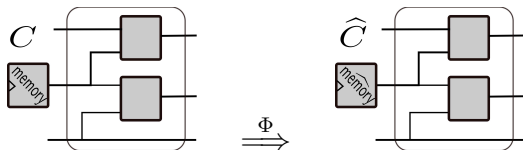
The core (red part)



- The core of \widehat{C} consists of k sub-circuits (same topology as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$
 - NAND $\Rightarrow \widehat{\text{NAND}}$ (see below)



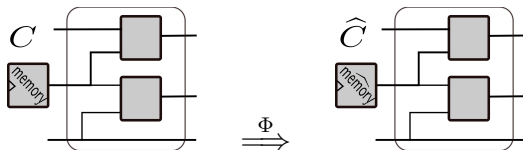
The core (red part)



- The core of \widehat{C} consists of k sub-circuits (**same topology** as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$
 - NAND $\Rightarrow \widehat{\text{NAND}}$ (see below)
 - **Valid** output of core: k copies of $\text{MMC}(w)$, $\forall w \in \text{output of } C$
($2k$ bits of randomness **in total**)

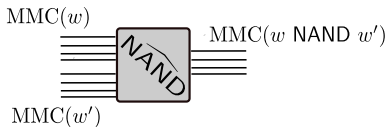


The core (red part)

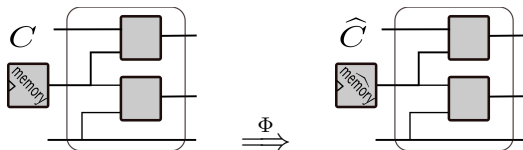


- The core of \widehat{C} consists of k sub-circuits (**same topology** as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$
 - NAND $\Rightarrow \widehat{\text{NAND}}$ (see below)
 - **Valid** output of core: k copies of $\text{MMC}(w)$, $\forall w \in \text{output of } C$ ($2k$ bits of randomness **in total**)

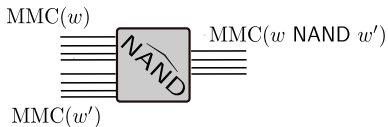
- Computes with MMC



The core (red part)



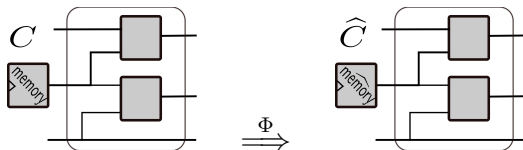
- The core of \widehat{C} consists of k sub-circuits (**same topology** as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$
 - NAND $\Rightarrow \widehat{\text{NAND}}$ (see below)
 - **Valid** output of core: k copies of $\text{MMC}(w)$, $\forall w \in \text{output of } C$ ($2k$ bits of randomness **in total**)



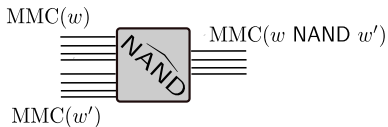
- Computes with MMC
- **Invalid** inputs generate 0^4



The core (red part)



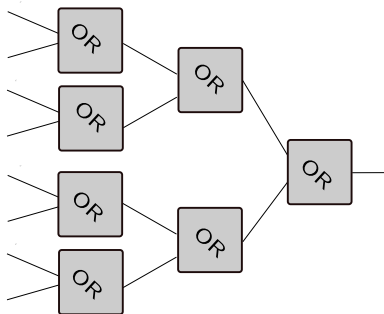
- The core of \widehat{C} consists of k sub-circuits (**same topology** as C)
 - A wire $w \in \{0, 1\} \Rightarrow \text{MMC}(w) = (w \oplus r, r, \bar{w} \oplus r', r')$
 - NAND $\Rightarrow \widehat{\text{NAND}}$ (see below)
 - **Valid** output of core: k copies of $\text{MMC}(w)$, $\forall w \in \text{output of } C$ ($2k$ bits of randomness **in total**)




- Computes with MMC
- **Invalid** inputs generate 0^4
- Assumed **tamper-proof**

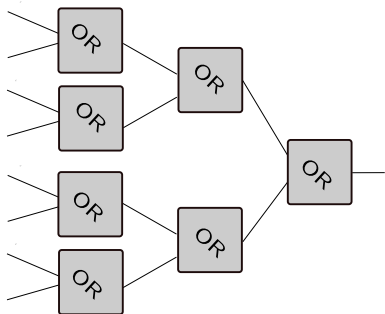


Why MMC?

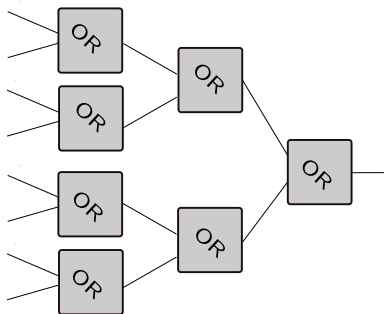



Why MMC?

- Say output is 0, i.e. all wires are 0
and  wants to change it to 1



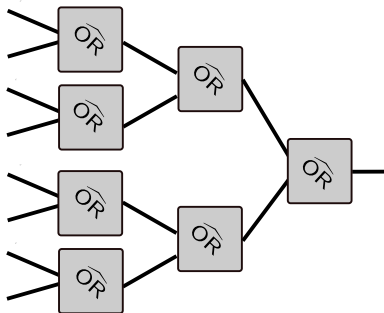
Why MMC?




- Say output is 0, i.e. all wires are 0 and  wants to change it to 1
- Just set **every** wire to 1: Prob. of success increases with # of wires!



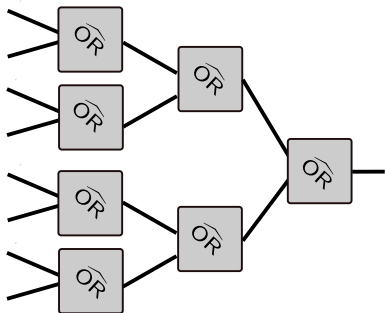
Why MMC?




- Say output is 0, i.e. all wires are 0 and  wants to change it to 1
- Just set **every** wire to 1: Prob. of success increases with # of wires!
- MMC prevents this attack: error will propagate!




Why MMC?



- Say output is 0, i.e. all wires are 0 and  wants to change it to 1
- Just set **every** wire to 1: Prob. of success increases with # of wires!
- MMC prevents this attack: error will propagate!
- Composition lemma: Tampering in a sub-circuit \Rightarrow output of core will contain **invalid** encoding w.h.p.

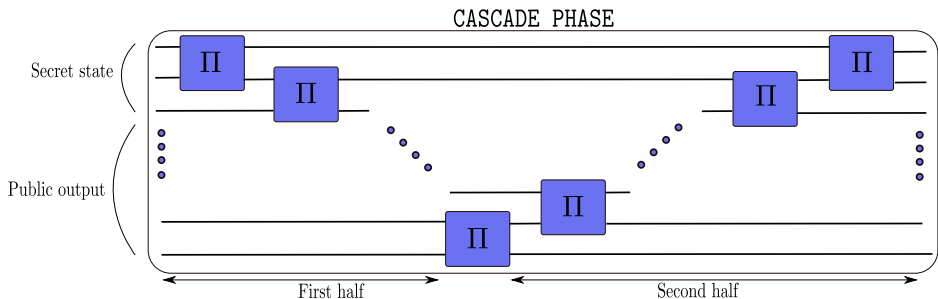



The cascade phase of [IPSW06]

- So changing the output of core will fail, but  can tamper over many rounds!

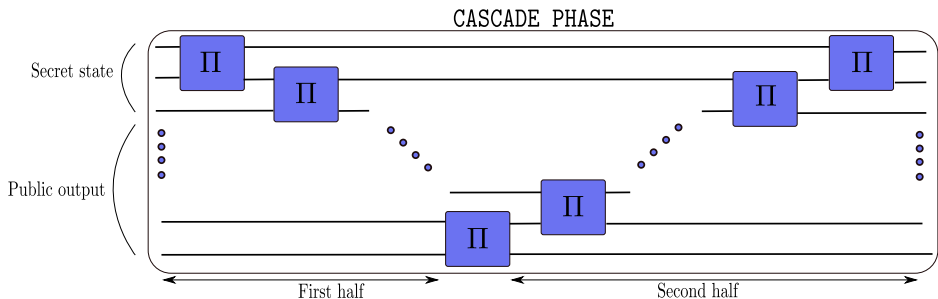



The cascade phase of [IPSW06]



- So changing the output of core will fail, but  can tamper over many rounds!
- Cascade phase will avoid this
 - Invalid input \Rightarrow output will encode 0: **self-destruct mechanism**

The cascade phase of [IPSW06]



- So changing the output of core will fail, but  can tamper over many rounds!
- Cascade phase will avoid this
 - Invalid input \Rightarrow output will encode 0: **self-destruct mechanism**
 - Tamper-proof gadgets of **linear size** (but **same** for every circuit)

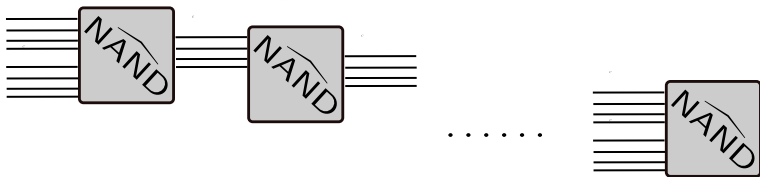
Why tamper-proof gadgets?

- We don't know how to prove without them 😞



Why tamper-proof gadgets?

- We don't know how to prove without them ☹️

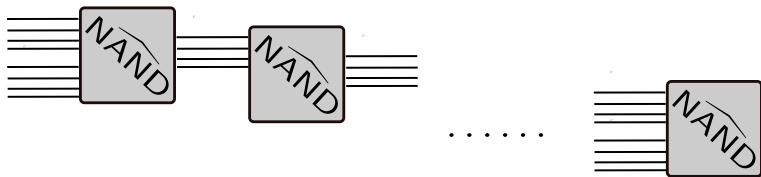



- Assume  can tamper inside the gadgets



Why tamper-proof gadgets?

- We don't know how to prove without them ☹

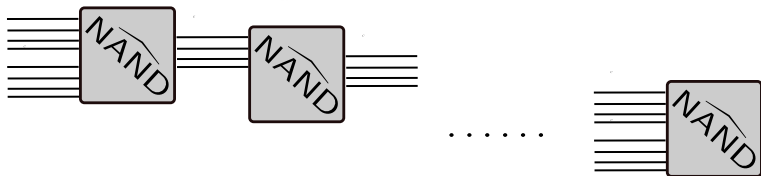



- Assume  can tamper inside the gadgets
 - Tampering with the input induces some distribution



Why tamper-proof gadgets?

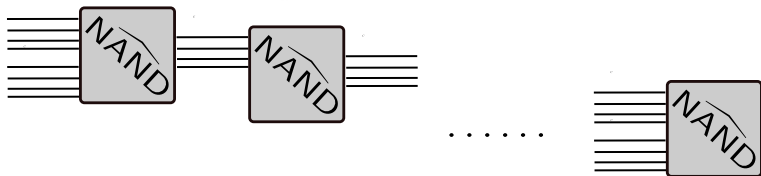
- We don't know how to prove without them ☹




- Assume  can tamper inside the gadgets
 - Tampering with the input induces some distribution
 - The deeper we go the “worse” this distribution can be made

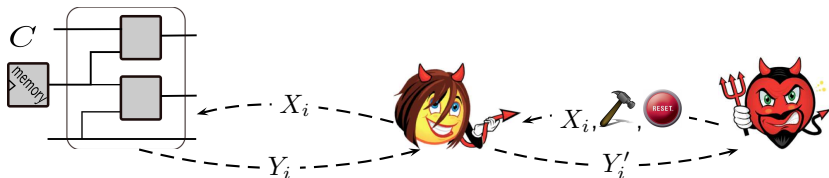
Why tamper-proof gadgets?

- We don't know how to prove without them ☹️

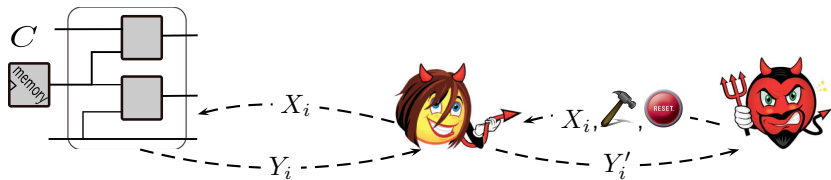



- Assume  can tamper inside the gadgets
 - Tampering with the input induces some distribution
 - The deeper we go the “worse” this distribution can be made
 - Open question: find a construction for the $\widehat{\text{NAND}}$ such that the bias cannot be increased

Proof sketch (1/2)



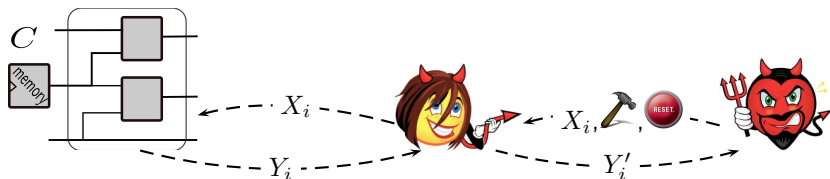
Proof sketch (1/2)




If  tampers with \hat{C} the following can happen



Proof sketch (1/2)

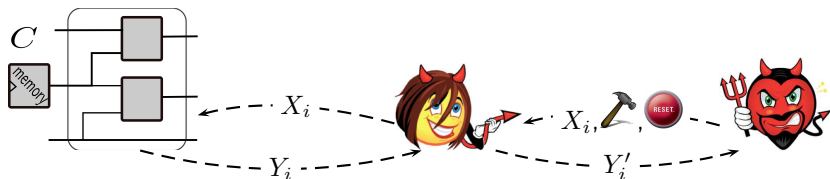



If  tampers with \hat{C} the following can happen

- 1 Tampering changes encoding of w to encoding of $1 - w$



Proof sketch (1/2)

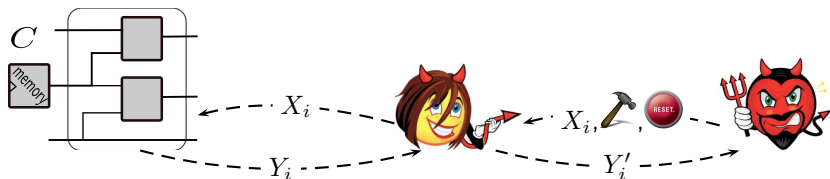



If  tampers with \hat{C} the following can happen

- ① Tampering changes encoding of w to encoding of $1 - w$
 - **Cannot** be simulated



Proof sketch (1/2)

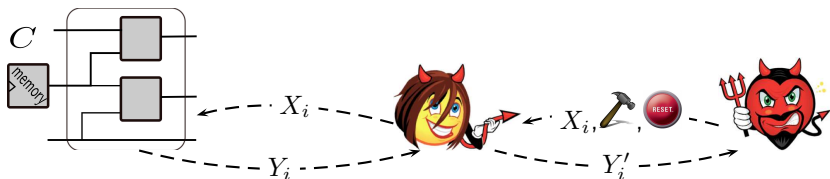



If  tampers with \hat{C} the following can happen

- 1 Tampering changes encoding of w to encoding of $1 - w$
 - **Cannot** be simulated
 - We show it happens with **negligible probability**



Proof sketch (1/2)

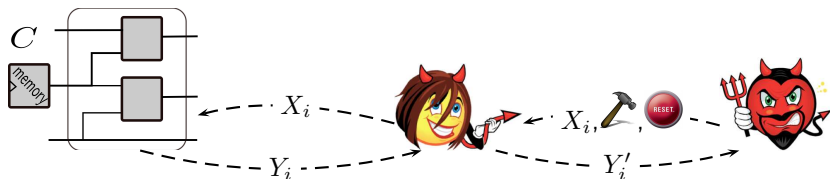



If  tampers with \hat{C} the following can happen

- 1 Tampering changes encoding of w to encoding of $1 - w$
 - **Cannot** be simulated
 - We show it happens with **negligible probability**
- 2 No tampering: use black box access for simulation



Proof sketch (1/2)

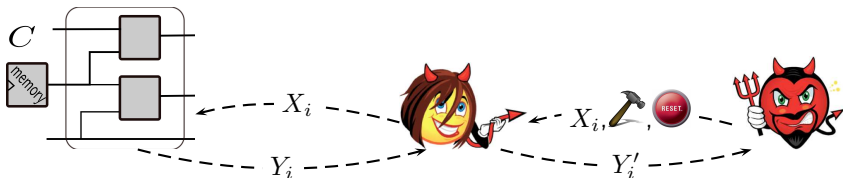


If  tampers with \hat{C} the following can happen

- 1 Tampering changes encoding of w to encoding of $1 - w$
 - **Cannot** be simulated
 - We show it happens with **negligible probability**
- 2 No tampering: use black box access for simulation
- 3 Tampering detected: output 0



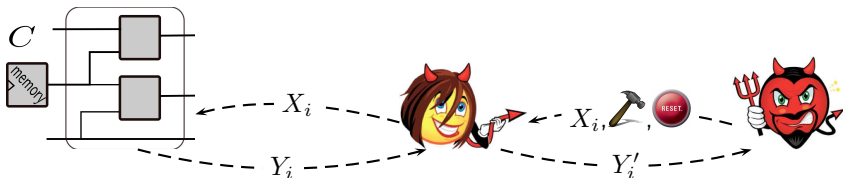
Proof sketch (2/2)




- However  does **not** know when this will happen



Proof sketch (2/2)




- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**



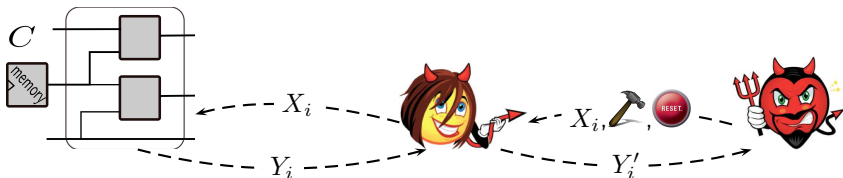
Proof sketch (2/2)




- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**
 - In which invocation



Proof sketch (2/2)




- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**
 - In which invocation
 - At which point of the cascade phase



Proof sketch (2/2)



- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**


$O(\log |M_0|)$ bits

- In which invocation
- At which point of the cascade phase



Proof sketch (2/2)



- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**


$O(\log |M_0|)$ bits

- In which invocation
- At which point of the cascade phase
- Finally, simulation must continue even **after** self-destruct



Proof sketch (2/2)



- However  does **not** know when this will happen
- Give as advice $\Lambda = f(M_0)$ the **exact point of failure**

$O(\log |M_0|)$ bits

- In which invocation
- At which point of the cascade phase
- Finally, simulation must continue even **after** self-destruct
 - Looks trivial since the state is destroyed, but recall that faults are **persistent**

Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
- 2 Trading a **small** amount of leakage can lead to **efficient** compilers



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
 - 2 Trading a **small** amount of leakage can lead to **efficient** compilers
- Where do we go from here?



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
 - 2 Trading a **small** amount of leakage can lead to **efficient** compilers
- Where do we go from here?
 - Dependent errors



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
 - 2 Trading a **small** amount of leakage can lead to **efficient** compilers
- Where do we go from here?
 - Dependent errors
 - Global tampering functions



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
 - 2 Trading a **small** amount of leakage can lead to **efficient** compilers
- Where do we go from here?
 - Dependent errors
 - Global tampering functions
 - Eliminate tamper-proof gadgets



Take-home message

- 1 It is possible to compile **any** circuit such that it resists an **unbounded** number of faults
 - 2 Trading a **small** amount of leakage can lead to **efficient** compilers
- Where do we go from here?
 - Dependent errors
 - Global tampering functions
 - Eliminate tamper-proof gadgets
 - Implementation-independent model



Questions?

THE END!

