: Versatile Software Obfuscation from a Lightweight Secure Element

Darius Mercadier[1] Viet Sang Nguyen[2] Matthieu Rivain[3] Aleksei Udovenko[4]

CHES 2024, September 6th

[1]Google
[2]Université Jean Monnet
[3]CryptoExperts
[4]SnT, University of Luxembourg

Acknowledgements:
French ANR SWITECH project (ANR-AAPG2019)
Luxembourg's FNR and Germany's DFG joint project APLICA (C19/IS/13641232)

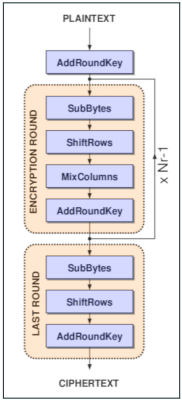
anr agence nationale de la recherche
AU SERVICE DE LA SCIENCE

Luxembourg National Research Fund

Plan

(Cryptographic) Code obfuscation

White-box cryptography



```
#include <stdio.h>
#include <math.h>
#define E return
#define S for
char*J="LJFFF%7544x^H^XXHZZ
I%KBPBEP%CBPEaIqBAI%CAaIqBq
AaIqB%AAaIqBEH%AAPBaIqB%PCD
%C@qJBH%AAaIqBAI%C@cJ%"    "
3P%H@ABhIaBBI%P@S@PC@",
typedef struct{x c,a,t;
;y G(x t,x a,x c){K.c=t    ;
nt)=putchar,X=40,z=5,o,   a
+c*b.a,b.t*c+a.t);}x H=   .
b.t+b.c*a.c+a.a*b.a;}x Q(){
a,pow(O(a,a),-H));}x D(y p)
++b],b<=r;Q())M=p.t?q -M PI
o+a,w=q*(w>t+H*a?o:    w>t?
1):A(M,p,U(O(A(P,M,T)    ,p
/8+8; M=Q      ()?o&1
int main(    int    L,char
++e)S(o=a    =0,j    =J+9;(c=
32,b++[B]    =G(q    +=*j/8&3
T:1), (c&    7)+    1e-4,o>2
?0:m(c),a          );*++j)==(
F<110;)S(L=-301;p=Z,++L<300
=G(-4,4.6,29),d=V(A(A(A(Z,V
(30.75,-6,-75),20)),g=R=255
A(h,d,i)));R=1<.01);S(N=V(A
U(i/3-D(A(h,N,i/3)))/pow(
M=V(G(T,1,2)),d,T))))
O(N,M))*H*Y+Y,g*=
q,q,1); P=A(p,s
);d=A(d,N,-2*O
```

Credits: Matt Zucker. IOCCC obfuscation contest 2011.

(Cryptographic) Code obfuscation

Generic code obfuscation

```c
#include <stdio.h>

int main(void)
{
  for(int i=1; i<=100; ++i)
  {
    if (i % 3 == 0)
      printf("Fizz");
    if (i % 5 == 0)
      printf("Buzz");
    if (i % 3 && i % 5)
      printf("%d", i);
    printf("\n");
  }

  return 0;
}
```

```c
#include <stdio.h>
#include <math.h>
#define E return
#define S for
char*J="LJFFF%7544x^H^XXHZZ
I%KBPBEP%CBPEaIqBAI%CAaIqBq
AaIqB%AAaIqBEH%AAPBaIqB%PCD
%C@qJBH%AAaIqBAI%C@cJ%"   "
JP%H@ABhIaBBI%P@S@PC#",
typedef struct{x c,a,t;
;y G(x t,x a,x c){K.c=t   ;
nt)=putchar,X=40,z=5,o,   a
+c*b.a,b.t*c+a.t);}x H=   .
b.t+b.c*a.c+a.a*b.a;}x Q(){
a,pow(O(a,a),-H));}x D(y p)
++b],b<=r;Q())M=p.t?q =M_PI
o+a,w=q*(w>t+H*a?o:   w>t?
1):A(M,p,U(O(A(P,M,T)   ,p
/8+8; M=Q         ()?o&1
int main(   int    L,char
++e)S(o=a   =0,j   =J+9;(c=
32,b++[B]   =G(q   +=*j/8&3
T:1), (c&   7)+    1e-4,o>2
70:m(c),a       ):*++j>=(
F<110;)S(L=-301;p=Z,++L<300
=G(-4,4.6,29),d=V(A(A(A(Z,V
(30.75,-6,-75),20)),g=R=255
A(h,d,i)));R=i<.01);S(N=V(A
U(i/3-D(A(h,N,i/3)))/pow(
M=V(G(T,1,2)),d,T))))
O(N,M))*H*Y+Y,g*=
q,q,1); p=A(p,s
);d=A(d,N,-2*O
```

Credits: Matt Zucker. IOCCC obfuscation contest 2011.

Obfuscation: State-of-the-Art

White-box Cryptography:

- Practical & Fast

  Chow, Eisen, Johnson, and Oorschot 2002

Theoretical Obfuscation (iO):

Obfuscation: State-of-the-Art

White-box Cryptography:

- Practical & Fast

  Chow, Eisen, Johnson, and Oorschot 2002

Theoretical Obfuscation (iO):

- Secure

  Jain, Lin, and Sahai 2021

Obfuscation: State-of-the-Art

White-box Cryptography:

- Practical & Fast

  Chow, Eisen, Johnson, and Oorschot 2002

- Totally insecure

  Billet, Gilbert, and Ech-Chatbi 2004

Theoretical Obfuscation (iO):

- Secure

  Jain, Lin, and Sahai 2021

Obfuscation: State-of-the-Art

White-box Cryptography:

- Practical & Fast

  Chow, Eisen, Johnson, and Oorschot 2002

- Totally insecure

  Billet, Gilbert, and Ech-Chatbi 2004

Theoretical Obfuscation (iO):

- Totally impractical

- Secure

  Jain, Lin, and Sahai 2021

Besides iO and White-box

Besides iO and White-box

Hardware security:

Besides iO and White-box

Hardware security:

Besides iO and White-box

Hardware security:

Besides iO and White-box

Hardware security:

Besides iO and White-box

Hardware security:

Trusted
Execution
Environment (TEE)
(SGX, TrustZone, ...)

Besides iO and White-box

Hardware security:

Increased functionality & complexity

Trusted
Execution
Environment (TEE)
(SGX, TrustZone, ...)

Besides iO and White-box

Hardware security:

Trusted Execution Environment (TEE) (SGX, TrustZone, ...)

Increased functionality & complexity

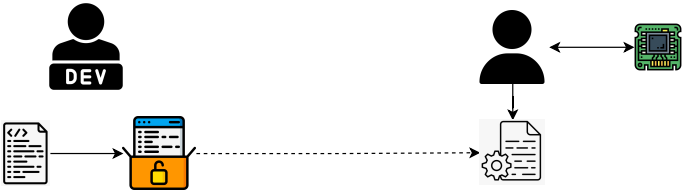
Larger attack surface

Obfuscating with Hardware

- "Founding Cryptography on Tamper-Proof Hardware Tokens"
  Goyal, Ishai, Sahai, Venkatesan, and Wadia 2010 TCC
- Program *obfuscation* using *stateless* secure HW tokens

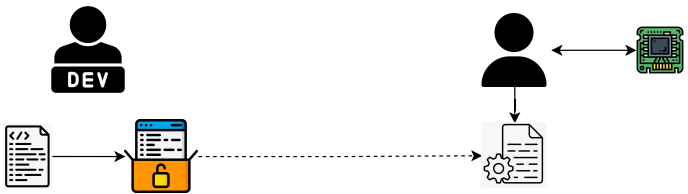
Obfuscating with Hardware

- "Founding Cryptography on Tamper-Proof Hardware Tokens"
  Goyal, Ishai, Sahai, Venkatesan, and Wadia 2010 TCC
- Program *obfuscation* using *stateless* secure HW tokens



Obfuscating with Hardware

- "Founding Cryptography on Tamper-Proof Hardware Tokens"
  Goyal, Ishai, Sahai, Venkatesan, and Wadia 2010 TCC
- Program *obfuscation* using *stateless* secure HW tokens



This work: exploring the design space, generalization

Focus: performance and user-friendliness (and security)

Our work - Overview

1. Compiler from a subset of C to an “obfuscated” bytecode

Our work - Overview

1. Compiler from a subset of C to an “obfuscated” bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms
4. Rectangular universalization for protecting logic/data dependencies

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms
4. Rectangular universalization for protecting logic/data dependencies
5. Applications: traceable white-box ciphers, neural networks, ...

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms
4. Rectangular universalization for protecting logic/data dependencies
5. Applications: traceable white-box ciphers, neural networks, ...

Increased functionality & complexity

Larger attack surface

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms
4. Rectangular universalization for protecting logic/data dependencies
5. Applications: traceable white-box ciphers, neural networks, ...

OBFUSCURO (NDSS'19) IRON (CCS'17)

SGX

Larger attack surface

Our work - Overview

1. Compiler from a subset of C to an "obfuscated" bytecode
2. Generalize instruction set: Boolean circuits $\rightarrow$ 32-bit instructions
3. Instruction batching (multi-instructions) and clusterization algorithms
4. Rectangular universalization for protecting logic/data dependencies
5. Applications: traceable white-box ciphers, neural networks, ...

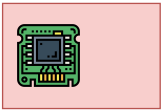
PHANTOM (CCS'13) | GhostRider (ASPLOS'15) | HOP (NDSS'17) | OBFUSCURO (NDSS'19) | IRON (CCS'17)

CPU + ORAM | SGX

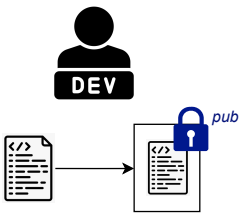
Larger attack surface

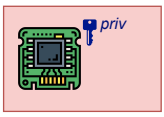
Plan

Obfuscation with Secure Element

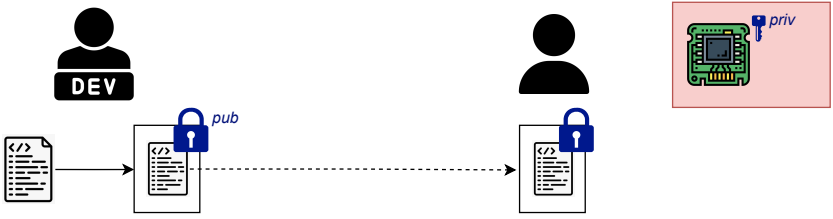


Obfuscation with Secure Element

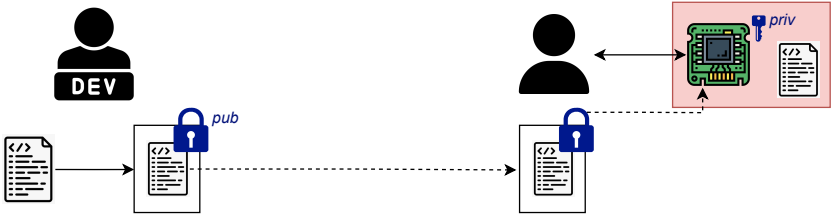




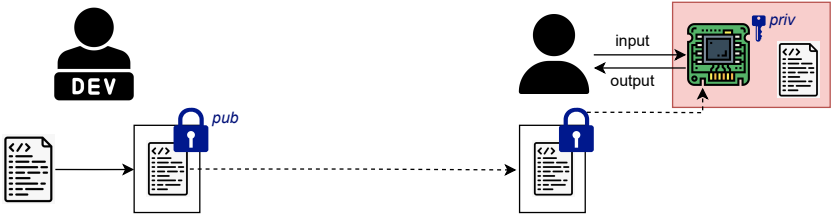
Obfuscation with Secure Element



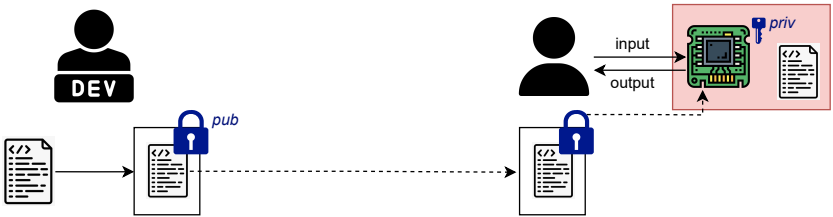
Obfuscation with Secure Element



Obfuscation with Secure Element

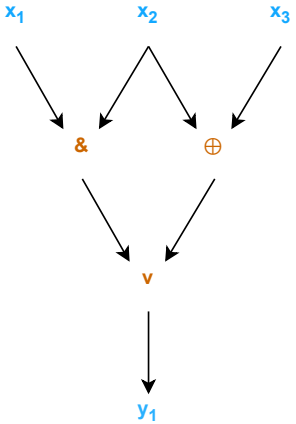


Obfuscation with Secure Element

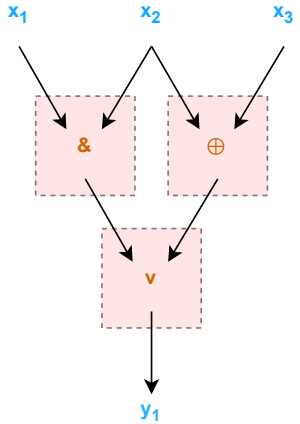


Problem: too complex secure element!

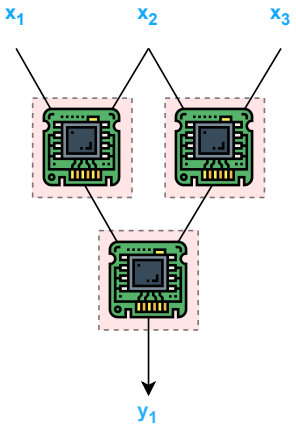
Scheme from TCC'10 (1/2)



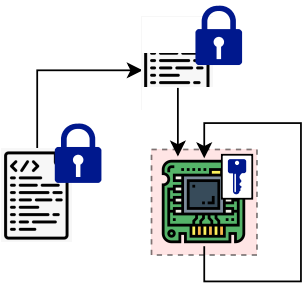
Scheme from TCC'10 (1/2)



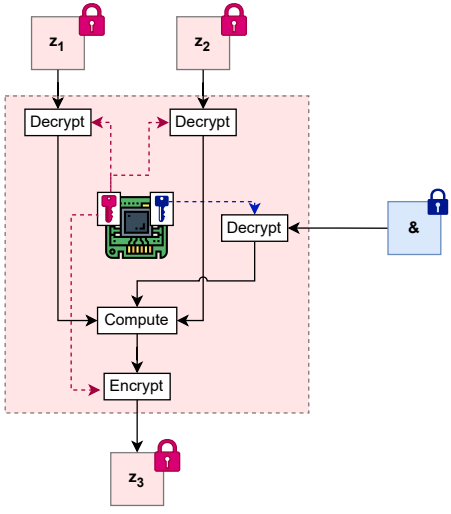
Scheme from TCC'10 (1/2)



Scheme from TCC'10 (1/2)



Scheme from TCC'10 (1/2)



Scheme from TCC'10 (2/2)

Prevent mix-and-match attacks:

- Authenticate node labels

Scheme from TCC'10 (2/2)

Prevent mix-and-match attacks:

- Authenticate node labels
- Pass around execution identity $\approx$ hash of the full input

Scheme from TCC'10 (2/2)

Prevent mix-and-match attacks:

- Authenticate node labels
- Pass around execution identity $\approx$ hash of the full input
- Lock to obfuscation identity

Scheme from TCC'10 (2/2)

Prevent mix-and-match attacks:

- Authenticate node labels
- Pass around execution identity $\approx$ hash of the full input
- Lock to obfuscation identity

Hybrid encryption:

- Decrypt symmetric key for instructions once, *reencrypt* using internal symm. key

Scheme from TCC'10 (2/2)

Prevent mix-and-match attacks:

- Authenticate node labels
- Pass around execution identity $\approx$ hash of the full input
- Lock to obfuscation identity

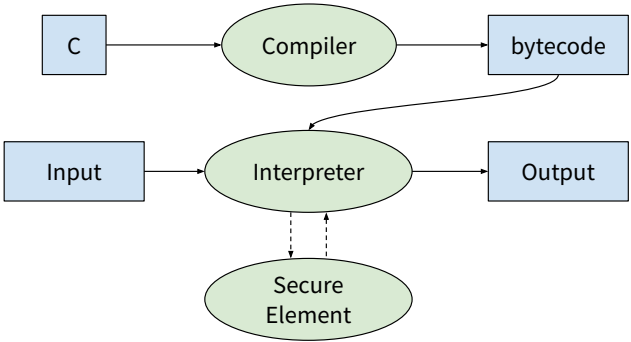
Hybrid encryption:

- Decrypt symmetric key for instructions once, *reencrypt* using internal symm. key

More details omitted, 4-5 different query types needed...

Plan

High-level overview



Example C code

```
unsigned int sum_naive(const unsigned int *array)
{
  // fix the size of array
  unsigned int n = 1000;
  unsigned int i;
  unsigned int s = 0;
  for (i=0; i<n; i++){
    s += array[i];
  }
  return s;
}
```

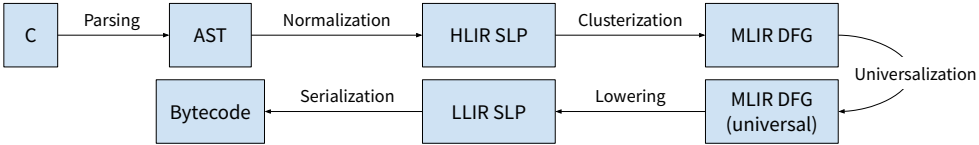
Example C code

```c
unsigned int sum_naive(const unsigned int *array)
{
  // fix the size of array
  unsigned int n = 1000;
  unsigned int i;
  unsigned int s = 0;
  for (i=0; i<n; i++){
    s += array[i];
  }
  return s;
}
```

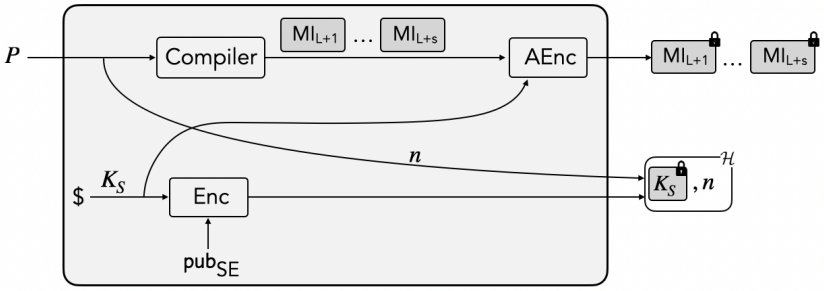
Subset of C language:

- only `uint32` (incl. pointers) supported
- constant-length loops (to be unrolled)
- no data-dependent control flow
- ternary operator allowed:
  `condition ? expr1 : expr2`

Compilation chain



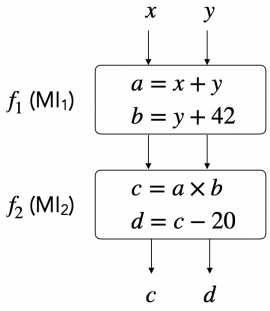
Obfuscation process



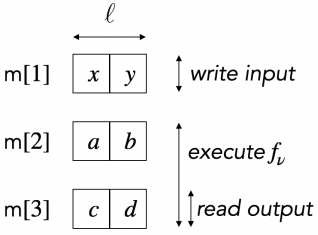
Multi-instructions

Inputs: x, y
Outputs: c, d
Body:
$a = x + y$
$b = y + 42$
$c = a \times b$
$d = c - 20$

Straightline Program

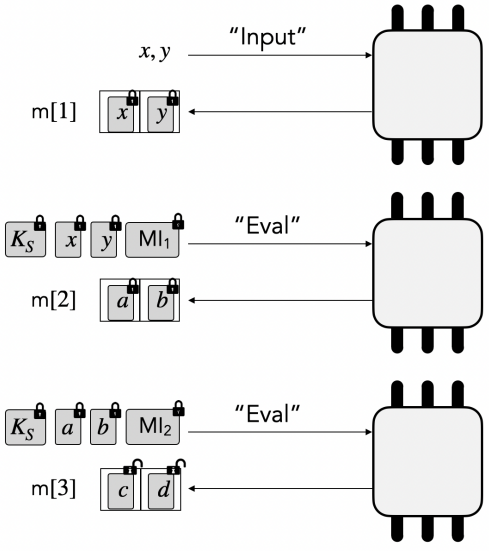


Multi-instructions

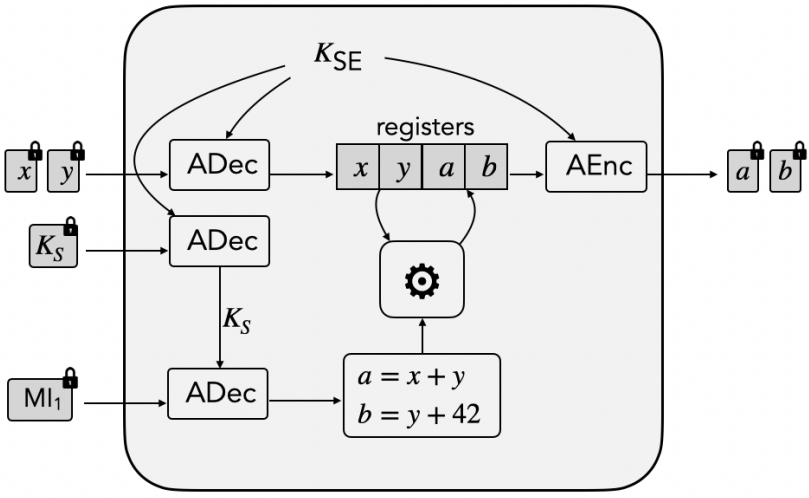


Memory

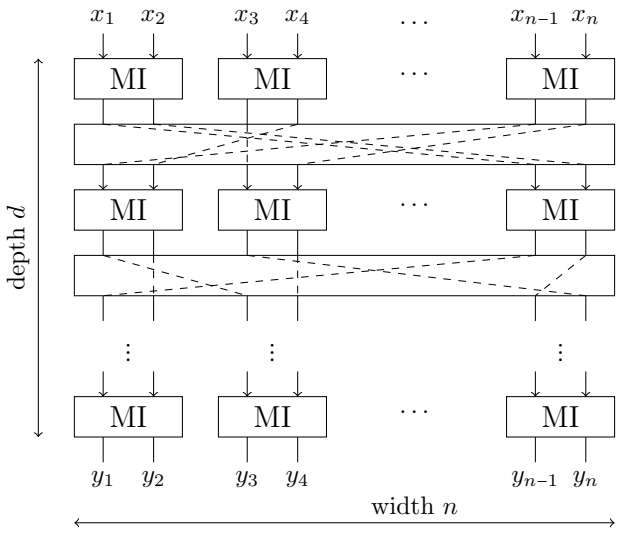
SE queries



SE queries


K_{SE}
registers
x
y
ADec
x y a b
AEnc
a
b
K_S
ADec
K_S
MI$_1$
ADec
$a = x + y$
$b = y + 42$

Universalization



Plan

SE parameters

SE name	#MI inputs &outputs	#MI instr.	Estimated performance on ARM Cortex-M3 (120 MHz)
small	8	32	600 MIs/sec
medium	16	64	300 MIs/sec
large	32	128	150 MIs/sec
extra-large	64	256	75 MIs/sec

Benchmarks - White-box mode

Source	Secure Element	#instr.	#MIs (final)	Compil. time	Exec. time (est.)
AES	small	5.3k	290	3.2 sec	0.5 sec
	medium		120	3.1 sec	0.4 sec
	large		59	3.1 sec	0.4 sec
	xlarge		29	3.2 sec	0.4 sec
Traceable AES	small	11k	580	4.8 sec	1.0 sec
	medium		240	4.4 sec	0.8 sec
	large		120	4.8 sec	0.8 sec
	xlarge		59	4.7 sec	0.8 sec
Neural Net	small	230k	22k	220 min	36.7 sec
	medium		11k	58 min	36.7 sec
	large		5.5k	21 min	36.7 sec
	xlarge		2.6k	520 sec	36.7 sec

White-box obfuscation mode. Time estimated on ARM Cortex-M3 120 MHz.

Benchmarks - Full obfuscation

Source	Secure Element	#instr.	Depth	Width	#MIs (final)	Exec. time (est)
AES	small	5.3k	190	7	12k	20 sec
	medium		110	3	3.4k	11 sec
	large		58	2	1.0k	7 sec
	xlarge		29	1	0.1k	2 sec
sum(tree)	small	1000	6	1.2k	28k	47 sec
	medium		3	63	3.1k	11 sec
	large		4	56	3.8k	26 sec
	xlarge		3	46	2.0k	27 sec
findmax(tree)	small	2k	5	190	24k	40 sec
	medium		3	63	3k	11 sec
	large		3	57	3k	20 sec
	xlarge		3	47	2k	27 sec

Full obfuscation mode.

Plan

Conclusions

- *Obfuscation* framework with provable reduction to HW security
  - Compilation from C programs
  - Rectangular universalization
  - Interpreter & runtime simulator

Conclusions

- *Obfuscation* framework with provable reduction to HW security
  - Compilation from C programs
  - Rectangular universalization
  - Interpreter & runtime simulator
- Stateless and lightweight HW requirement: reduced attack surface

(?) *Open question:* protected hardware design

Conclusions

- *Obfuscation* framework with provable reduction to HW security
  - Compilation from C programs
  - Rectangular universalization
  - Interpreter & runtime simulator
- Stateless and lightweight HW requirement: reduced attack surface

(?) *Open question:* protected hardware design

github.com/CryptoExperts/OBSCURE

tches.iacr.org/index.php/TCHES/article/view/11440