Tales of software protection

Cryptography and obfuscation, better together

About

Arnau Gàmez i Montolio

Hacker, Reverse Engineer & Mathematician

Occupation

- Senior Expert Engineer, Security @ Activision-Blizzard
- Founder, Researcher & Trainer @ Fura Labs
- PhD @ City, UoL

Social

- Tweets: @arnaugamez, @furalabs
- Toots: {@arnaugamez, @furalabs}@infosec.exchange

Context

Technical protection against Man-At-The-End (MATE) attacks.

The attacker/analyst has an instance of the program and completely controls the environment where it is executed.

Protection against end-users				
Legal	Technical			
	Obfuscation	*Encryption*	Server-side execution	Trusted native code

Obfuscation

Transform an input program P into a functionally equivalent program P' which is harder to analyze and extract information from than P.

$$P \longrightarrow \boxed{\text{Obfuscation}} \longrightarrow P'$$

Obfuscation

Transform an input program P into a functionally equivalent program P' which is harder to analyze and extract information from than P.

$$P \longrightarrow \boxed{\text{Obfuscation}} \longrightarrow P'$$

Motivation

~~Prevent~~ complicate reverse engineering.

Presence

Presence

Commercial software protection:

- Intellectual property
- Digital Rights Management (DRM)
- (Anti-)cheating

Presence

Commercial software protection:

- Intellectual property
- Digital Rights Management (DRM)
- (Anti-)cheating

Malware threats:

- Avoid automatic signature detection
- Slow down analysis → time → money

Methodology

Apply semantics-preserving transformations to program's control-flow and data-flow.

Methodology

Apply semantics-preserving transformations to program's control-flow and data-flow.

- At different abstraction levels: source code, intermediate representation, compiled binary

Methodology

Apply semantics-preserving transformations to program's control-flow and data-flow.

- At different abstraction levels: source code, intermediate representation, compiled binary
- At different target units: whole program, function, basic block, instruction

Deobfuscation

Transform an obfuscated (piece of) program P' into a (piece of) program P'' which is easier to analyze and extract information from than P'.

$$P'' \longleftarrow \boxed{\text{Deobfuscation}} \longleftarrow P'$$

Deobfuscation

Transform an obfuscated (piece of) program P' into a (piece of) program P'' which is easier to analyze and extract information from than P'.

$$P'' \longleftarrow \boxed{\text{Deobfuscation}} \longleftarrow P'$$

Ideally $P'' \approx P$, but this is rarely the case

- Lack of access to original program P
- Interest in specific parts rather than whole program
- Interest in understanding rather than rebuilding

Mixed Boolean-Arithmetic (MBA)

Algebraic expressions composed of integer arithmetic operators ($+$, $-$, $\times$) and bitwise operators ($\wedge$, $\vee$, $\oplus$, $\neg$).

Mixed Boolean-Arithmetic (MBA)

Algebraic expressions composed of integer arithmetic operators ($+$, $-$, $\times$) and bitwise operators ($\wedge$, $\vee$, $\oplus$, $\neg$).

Notation reminder

Operation	Math	Code
AND	$\wedge$	&
OR	$\vee$	|
XOR	$\oplus$	^
NOT	$\neg$	~

Polynomial MBA expressions

$$43(x \wedge y \vee z)^2((x \oplus y) \wedge z \vee t) + 2x + 123(x \vee y)zt^2$$

Polynomial MBA expressions

$$43(x \wedge y \vee z)^2((x \oplus y) \wedge z \vee t) + 2x + 123(x \vee y)zt^2$$

Think of them as *normal* multivariate polynomials.

$$43X_1^2X_2 + 2X_3 + 123X_4X_5X_6^2$$

Polynomial MBA expressions

$$43(x \wedge y \vee z)^2((x \oplus y) \wedge z \vee t) + 2x + 123(x \vee y)zt^2$$

Think of them as *normal* multivariate polynomials.

$$43X_1^2X_2 + 2X_3 + 123X_4X_5X_6^2$$

But now, the variables are boolean expressions.

$$\begin{aligned} X_1 &= (x \wedge y \vee z) \\ X_2 &= ((x \oplus y) \wedge z \vee t) \\ X_3 &= x \\ X_4 &= (x \vee y) \\ X_5 &= z \\ X_6 &= t \end{aligned}$$

Linear MBA expressions

A subset of polynomial MBA expressions, where **only one boolean expression** is allowed per (summation) term.

$$(x \oplus y) + 2(x \wedge y)$$

Linear MBA expressions

A subset of polynomial MBA expressions, where **only one boolean expression** is allowed per (summation) term.

$$(x \oplus y) + 2(x \wedge y)$$

Again, it can be seen as $X_1 + 2X_2$, where $X_1 = (x \oplus y)$ and $X_2 = (x \wedge y)$.

Linear MBA expressions

A subset of polynomial MBA expressions, where **only one boolean expression** is allowed per (summation) term.

$$(x \oplus y) + 2(x \wedge y)$$

Again, it can be seen as $X_1 + 2X_2$, where $X_1 = (x \oplus y)$ and $X_2 = (x \wedge y)$.

Note: In practice, you can vaguely think of *linearity* as a restriction not allowing variables (in our case, boolean expressions) to end up being multiplied together.

Fact

State-of-the-art software protection mechanisms leverage MBA transformations to obfuscate code.

Why?

Why?

Combinations of operators from these different fields *do not interact well together*

- No general rules (distributivity, factorization...) or theory
- Computer algebra systems do not support bitwise operators with symbolic variables

Why?

Combinations of operators from these different fields *do not interact well together*

- No general rules (distributivity, factorization...) or theory
- Computer algebra systems do not support bitwise operators with symbolic variables

SMT solvers support for mixing operators (*bit vector* theory)

- In theory, reasonably good at proving semantic equivalence
    - Easily thwarted with deliberate MBA transformations
- Pretty bad at simplification for general MBA expressions

Obfuscate expressions

Rewrite rules

A chosen operator is rewritten with an equivalent MBA expression.

$$x + y \rightarrow (x \oplus y) + 2(x \wedge y)$$

Obfuscate expressions

Rewrite rules

A chosen operator is rewritten with an equivalent MBA expression.

$$x + y \rightarrow (x \oplus y) + 2(x \wedge y)$$

It is *easy* to generate arbitrary linear rewrite rules: there is a known constructive method.

These rewrite rules can be applied iteratively (thanks to composability of polynomial MBA expressions).

These rewrite rules can be applied iteratively (thanks to composability of polynomial MBA expressions).

Composability

Let

$$E_1(x, y), E_2(x, y), E_3(x, y)$$

be polynomial MBA expressions.

These rewrite rules can be applied iteratively (thanks to composability of polynomial MBA expressions).

Composability

Let

$$E_1(x, y), E_2(x, y), E_3(x, y)$$

be polynomial MBA expressions.

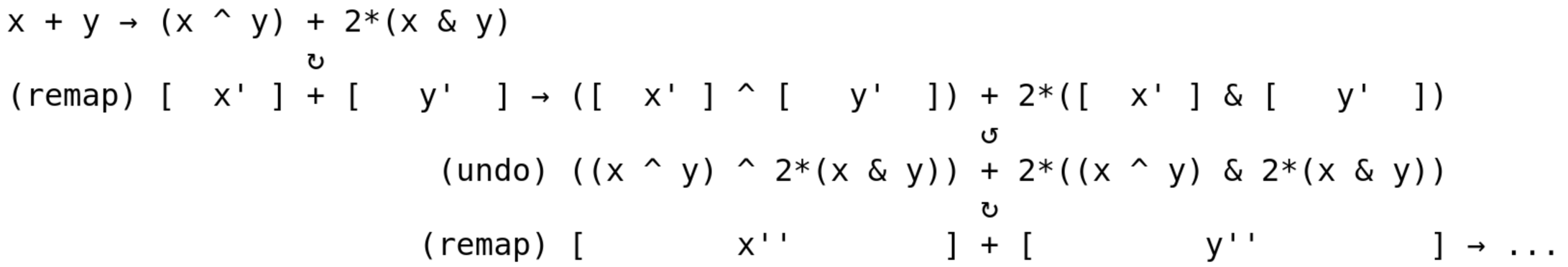
Then,

$$E_1(E_2(x, y), E_3(x, y))$$

is a polynomial MBA expression.

This can be seen as applying a rewrite rule consecutively, with an intermediate variable remap.

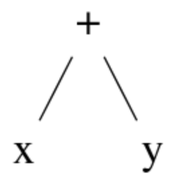
This can be seen as applying a rewrite rule consecutively, with an intermediate variable remap.



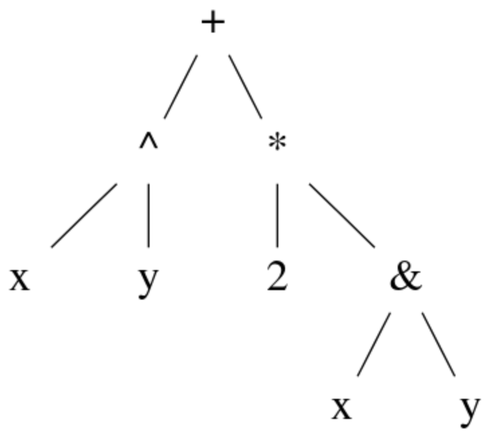
It can also be seen as rewrite rules that apply at the expressions' AST representation.

It can also be seen as rewrite rules that apply at the expressions' AST representation.

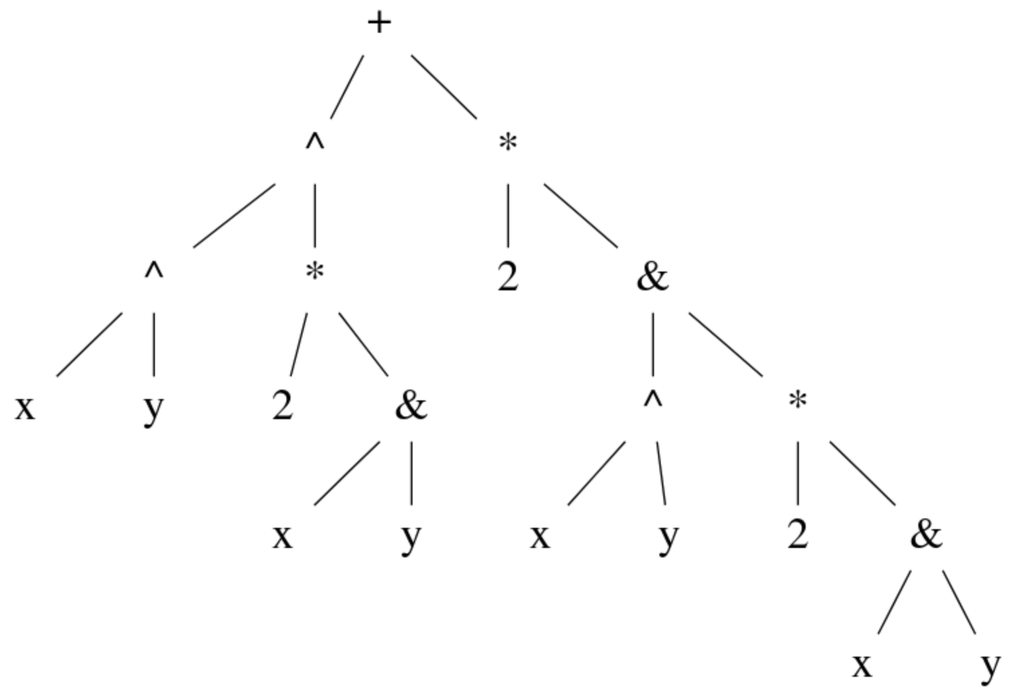
In other words, we can apply such operator rewrites to whatever subexpression is found at the *left* and *right* branches of its AST representation.



x + y



(x ^ y) + 2*(x & y)



((x ^ y) ^ 2*(x & y)) + 2*((x ^ y) & 2*(x & y))

Obfuscate expressions

Insertion of identities

Let e be the target expression, and consider $\{ f, f^{-1} \}$ to be a pair of inverse functions on n-bits.

We can rewrite e as $f^{-1}(f(e))$

$$
\begin{aligned}
e &= (x \oplus y) + 2 \times (x \wedge y) \\
f &: x \mapsto 39x + 23 \\
f^{-1} &: x \mapsto 151x + 111
\end{aligned}
$$

$$
\begin{aligned}
e &= (x \oplus y) + 2 \times (x \wedge y) \\
f &: x \mapsto 39x + 23 \\
f^{-1} &: x \mapsto 151x + 111
\end{aligned}
$$

$$
f^{-1}(f(e)) = 151 \times (39 \times ((x \oplus y) + 2 \times (x \wedge y)) + 23) + 111
$$

The function f is often an affine function (or a permutation polynomial).

Obfuscate constants

- K constant
- P, Q inverse permutation polynomials
- E non-trivially equal to zero MBA expression

Conceal constant: $K \equiv P(E + Q(K))$

$$
\begin{aligned}
K &= 123 \\
P(X) &= 97X + 248X^2 \\
Q(X) &= 161X + 136X^2 \\
E(x, y) &= x - y + 2(\neg x \wedge y) - (x \oplus y)
\end{aligned}
$$

$$
\begin{aligned}
K &= 123 \\
P(X) &= 97X + 248X^2 \\
Q(X) &= 161X + 136X^2 \\
E(x, y) &= x - y + 2(\neg x \wedge y) - (x \oplus y)
\end{aligned}
$$

$$
\begin{aligned}
P(E + Q(K)) = \; & 195+ \\
& 97x+ \\
& 159y+ \\
& 194\neg(x \vee \neg y)+ \\
& 159(x \oplus y) + (163 + x + 255y + 2\neg(x \vee \neg y) + 255(x \oplus y))\times \\
& (232 + 248x + 8y + 240\neg(x \vee \neg y) + 8(x \oplus y))
\end{aligned}
$$

Mixing operators in obfuscation vs cryptography

Mixing operators in obfuscation vs cryptography

Obfuscation

Given an initial expression E_1, generate a syntactically more complex expression E_2 which preserves the (simple) semantic behavior.

Mixing operators in obfuscation vs cryptography

Obfuscation

Given an initial expression E_1, generate a syntactically more complex expression E_2 which preserves the (simple) semantic behavior.

Cryptography

Mixing operators to achieve a semantically complex cryptosystem that verifies some properties (non-linearity, high algebraic degree) to guarantee confusion and diffusion.

	Obfuscation	Cryptography
Syntax	Complex	Complex
Semantics	Simple	Complex

Hardening cryptography with obfuscation

Hardening cryptography with obfuscation

Disclaimer: Of course there is an efficiency compromise when obfuscating *anything*.

Conceal recognizable computations of cryptographic algorithms

Goal

Difficult the recognition of a cryptosystem by examining its assembly or even (decompiled) source code.

Conceal recognizable computations of cryptographic algorithms

Goal

Difficult the recognition of a cryptosystem by examining its assembly or even (decompiled) source code.

Method

Leverage rewrite rules and insertion of identities to obfuscate the computations done by the cryptographic algorithm.

Tip: use different obfuscation transformations for repeating computations.

SHA-256 compression loop

```
#define Ch(x, y, z) ((x & y) ^ (~x & z))
#define Maj(x, y, z) ((x & y) ^ (x & z) ^ (y & z))
#define S0(x) (CYCLIC (x, 2) ^ CYCLIC (x, 13) ^ CYCLIC (x, 22))
#define S1(x) (CYCLIC (x, 6) ^ CYCLIC (x, 11) ^ CYCLIC (x, 25))
#define R0(x) (CYCLIC (x, 7) ^ CYCLIC (x, 18) ^ (x >> 3))
#define R1(x) (CYCLIC (x, 17) ^ CYCLIC (x, 19) ^ (x >> 10))
#define CYCLIC(w, s) ((w >> s) | (w << (32 - s)))

[...]

for (unsigned int t = 0; t < 64; ++t)
    {
      uint32_t T1 = h + S1 (e) + Ch (e, f, g) + K[t] + W[t];
      uint32_t T2 = S0 (a) + Maj (a, b, c);
      [...]
    }
```

```
#define Ch(x, y, z) ((x & y) ^ (~x & z))
```

```
#define Ch(x, y, z) ((x & y) ^ (~x & z))
```

```
#define Ch(x, y, z) (x + y - (x ^ y) - (x & y)) ^ ((x | z) - x)
```

```
#define Ch(x, y, z) ((x & y) ^ (~x & z))
```

```
#define Ch(x, y, z) (x + y - (x ^ y) - (x & y)) ^ ((x | z) - x)
```

```
#define Ch(x, y, z) (x + y - (x + y - 2*(x & y)) - (x & y)) + ((x | z) - x) - 2*((x
y) & ((x | z) - x))
```

```
uint32_t T1 = h + S1 (e) + Ch (e, f, g) + K[t] + W[t];
```

```
uint32_t T1 = h + S1 (e) + Ch (e, f, g) + K[t] + W[t];
```

```
uint32_t T1 = ((h + S1 (e)) | (Ch (e, f, g) + K[t] + W[t])) + (~(h + S1 (e)) | (Ch (
f, g) + K[t] + W[t])) - ~(h + S1 (e));
```

```
uint32_t T1 = h + S1 (e) + Ch (e, f, g) + K[t] + W[t];
```

```
uint32_t T1 = ((h + S1 (e)) | (Ch (e, f, g) + K[t] + W[t])) + (~(h + S1 (e)) | (Ch (
f, g) + K[t] + W[t])) - ~(h + S1 (e));
```

```
uint32_t T1 = ((((h + S1 (e)) | (Ch (e, f, g) + K[t] + W[t]))) ^ ((~(h + S1 (e)) | (
(e, f, g) + K[t] + W[t])) - ~(h + S1 (e)))) + 2*((((h + S1 (e)) | (Ch (e, f, g) + K[
+ W[t]))) & ((~(h + S1 (e)) | (Ch (e, f, g) + K[t] + W[t])) - ~(h + S1 (e))));
```

Conceal known constants of cryptographic algorithms

Goal

Thwart discovery of cryptographic algorithms by locating and cross-referencing known cryptosystem constants (FindCrypt).

Conceal known constants of cryptographic algorithms

Goal

Thwart discovery of cryptographic algorithms by locating and cross-referencing known cryptosystem constants (FindCrypt).

Method

Replace lookup of constants by MBA opaque constant expressions that compute them at runtime.

Tip: different expressions can be generated for every use of a constant.

SHA-256 initial hash values

```
0x6A09E667,0xBB67AE85,0x3C6EF372,0xA54FF53A,
0x510E527F,0x9B05688C,0x1F83D9AB,0x5BE0CD19
```

MD5 initial variables

```
0x67452301,0xEFCDAB89,0x98BADCFE,0x10325476
```

AES S-Box

```
0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD7,0xAB,0x76,...
```

White-box cryptography

Goal

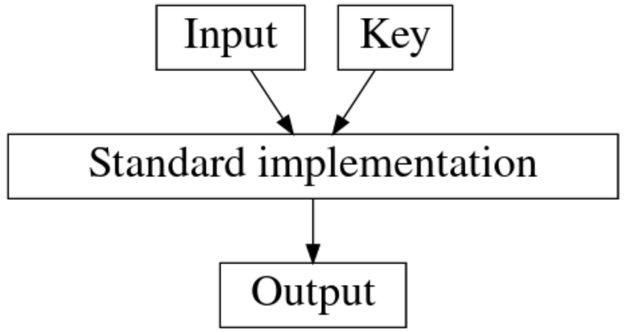
Prevent the extraction of embedded secret keys.

Method

Blend together the key and the cryptosystem algorithm in a unique implementation that embeds the key within the encryption/decryption computation itself.

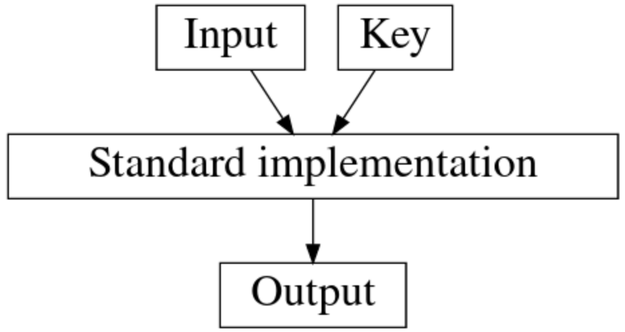
Method

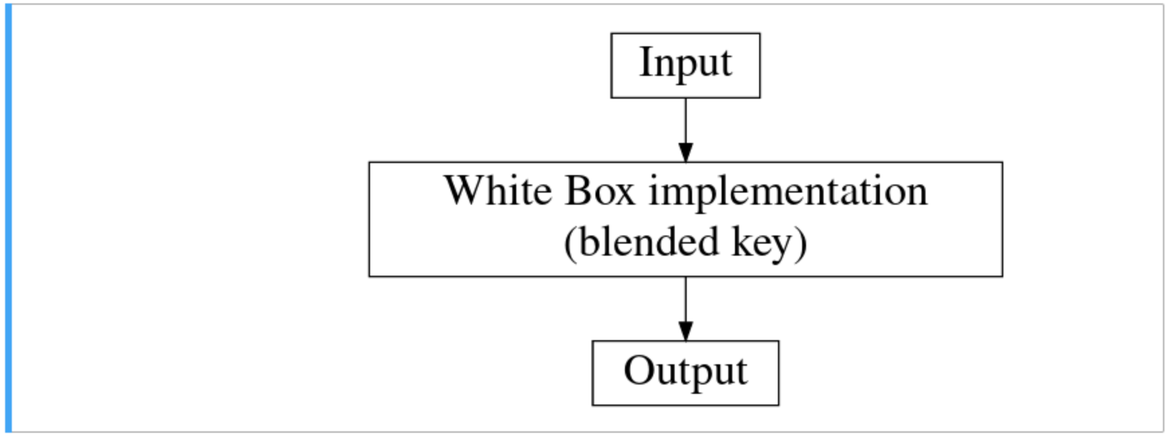
Blend together the key and the cryptosystem algorithm in a unique implementation that embeds the key within the encryption/decryption computation itself.



Method

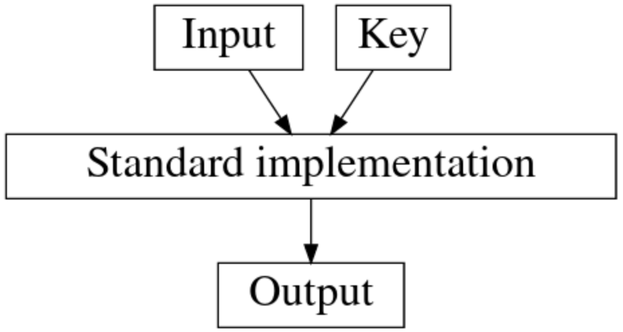
Blend together the key and the cryptosystem algorithm in a unique implementation that embeds the key within the encryption/decryption computation itself.

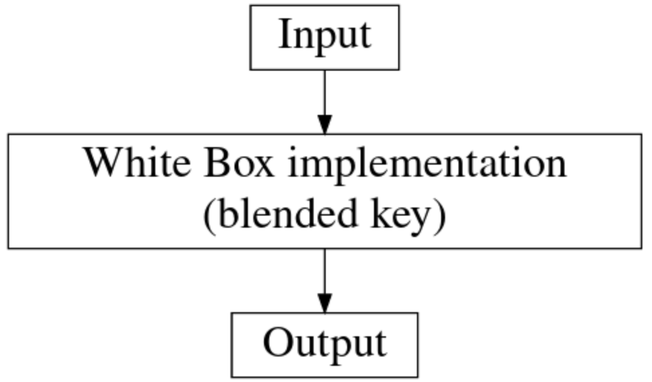




Method

Blend together the key and the cryptosystem algorithm in a unique implementation that embeds the key within the encryption/decryption computation itself.





Idea: MBA transformations provide a really nice set of primitives to develop custom white-box cryptography implementations.

Conclusions

Context matters. A lot.

Context matters. A lot.

- Who is your adversary and what are their capabilities?

Context matters. A lot.

- Who is your adversary and what are their capabilities?
- What do you need to protect?
  - Algorithms?
  - Constants?
  - Keys?

Context matters. A lot.

- Who is your adversary and what are their capabilities?
- What do you need to protect?
  - Algorithms?
  - Constants?
  - Keys?
- How much extra computational power can you afford?
  - For each asset?
  - At each program point?

Mixed Boolean-Arithmetic algebraic transformations are really powerful.

Mixed Boolean-Arithmetic algebraic transformations are really powerful.

- Not a lot of people knows about them (well)

Mixed Boolean-Arithmetic algebraic transformations are really powerful.

- Not a lot of people knows about them (well)

- As of today, virtually no commercial product uses them (correctly)
  - A set of very limited, only linear and copy-pasted rules used *everywhere*
  - Identities are mostly affine, or a very limited set of permutation polynomials.

Actual cryptographic implementations can be protected leveraging MBA transformations

Actual cryptographic implementations can be protected leveraging MBA transformations

- Algorithms, constants, keys

Actual cryptographic implementations can be protected leveraging MBA transformations

- Algorithms, constants, keys
- Use your imagination

More (crazy) ideas

More (crazy) ideas

- Control over syntactic complexity of operations
    - Certain control over *timing*
        - Manipulation of the duration of cryptographic operations
            - Dynamic, modify every run/trace/minute

More (crazy) ideas

- Control over syntactic complexity of operations
  - Certain control over *timing*
    - Manipulation of the duration of cryptographic operations
      - Dynamic, modify every run/trace/minute

...

More (crazy) ideas

- Control over syntactic complexity of operations
  - Certain control over *timing*
    - Manipulation of the duration of cryptographic operations
      - Dynamic, modify every run/trace/minute

...

Timing-based side-channel *honey pot*

Training

An analytical apporach to modern binary deobfuscation

A curated training that teaches you to build, analyze and defeat obfuscated code

- Public / Private
- In-person / Remote
- 4 days (flexible)
- Details: https://furalabs.com/trainings

Upcoming

- August 05-08, 2023 @ RingZer0 - Las Vegas (USA)
- August 21-24, 2023 @ HITB Asia - Phuket (Thailand)

Thank you

Q&A