White-box cryptography

Dmitry Khovratovich

University of Luxembourg

29 December 2013

This is a survey talk, partially inspired by the ongoing research on white-box cryptographic designs at the University of Luxembourg (together with Alex Biryukov).

`https://www.cryptolux.org`

White-box cryptography, what is it?

White-box cryptography, what is it?

- Obfuscation?

White-box cryptography, what is it?

- Obfuscation?
- Public-key cryptography?

White-box cryptography, what is it?

- Obfuscation?
- Public-key cryptography?
- No one knows exactly?

White-box cryptography, what is it?

- Obfuscation?
- Public-key cryptography?
- No one knows exactly?

All versions are partly true.

Overview

1. Man at the end
   - Standard model of adversary
   - Man at the end
   - Details of protection schemes
   - Key hiding problem
2. White-Box cryptography
   - White-box implementation
   - Lookup table method
   - White-boxing AES
   - Cryptanalysis
   - Market
   - Weak and strong white-box implementations
3. New approaches
   - WBC from scratch
   - Weak white-box candidate
   - Non-invertibility problem
   - White-box cryptography from polynomials

We consider protection of large amounts of data: databases, digital media, scientific experiments, etc..

We consider protection of large amounts of data: databases, digital media, scientific experiments, etc..

Performance is important

Performance and security are typically addressed with symmetric cryptography, where secret components are shared by parties.

Confidentiality/privacy:

- Symmetric ciphers: AES-128/192/256 (key size in bits).
- Modes of operation to process large data: CTR (counter), CBC (chaining).

Integrity/authenticity:

- Hash functions: SHA-1, SHA-256, SHA-3 (Keccak) and Message Authentication Codes (HMAC).
- Authenticated encryption: OCB, GCM.

Performance and security are typically addressed with symmetric cryptography, where secret components are shared by parties.

Confidentiality/privacy:

- Symmetric ciphers: AES-128/192/256 (key size in bits).
- Modes of operation to process large data: CTR (counter), CBC (chaining).

Integrity/authenticity:

- Hash functions: SHA-1, SHA-256, SHA-3 (Keccak) and Message Authentication Codes (HMAC).
- Authenticated encryption: OCB, GCM.

All these designs withstood years of cryptanalysis, and their security is often backed up with security proofs and arguments.

Symmetric vs. public-key cryptography

Performance has some cost:

- RSA is believed to be secure, because it is related to the hardness of factoring;
- AES is believed to be secure, just because no one has broken it (and many tried).

Standard model of adversary

Standard cryptographic model (dates back to Kerchoffs):

- Algorithm is open and available for public scrutiny;
- Key is secret (hidden in hardware, remote computer, etc.).

Sometimes the latter assumption does not hold.

Man at the end: unorthodox model for cryptographic algorithms

An attacker downloads a DRM-protected song from a server to an authorized music player:

- Connection can be eavesdropped;
- The player code is accessible;
- Dynamic execution allows to view the entire cryptographic transformation.
- Keys are difficult to hide.

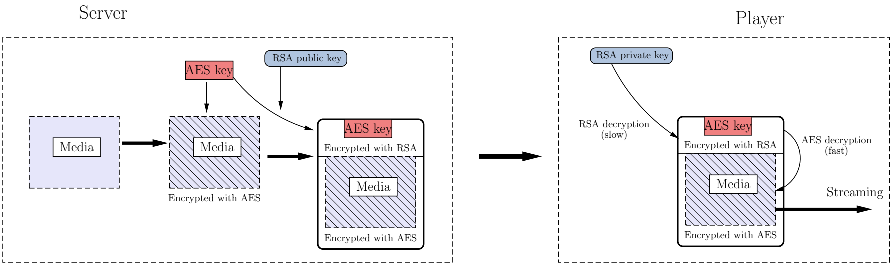
More applications:

- Distribution of digital cinema to theaters;
- Playing encrypted HD movies in authorized players (AACS for HD DVD).
- Direct streaming of protected media (DTCP).

Quite many protocols have been attacked. How?

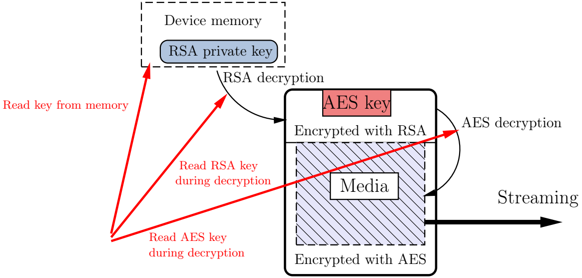
Details

How digital cinema is protected:



- Movie is encrypted with AES (since it is fast);
- AES key is encrypted with the RSA key of the device;
- To reduce the server workload the AES key may repeat.

Where it can be attacked:



CSS (1999), AACS (2007), HDCP (2010) have been broken along these lines.

Key or entire code

Quite often, it is possible to extract the entire code and use it for decryption (code lifting).

However, it is better to obtain keys directly:

- Easy to distribute;
- Easy to update;
- Not traceable;
- Sometimes the decoding procedure can not be isolated.

Quite often, it is possible to extract the entire code and use it for decryption (code lifting).

However, it is better to obtain keys directly:

- Easy to distribute;
- Easy to update;
- Not traceable;
- Sometimes the decoding procedure can not be isolated.

Staying apart from code lifting, how can we hide these keys?

Hardware (various tamper-proof dongles, TPM):

- Significantly increases the attack cost;
- Also more expensive, less flexible, and vulnerable to side-channel attacks (timing and power analysis).

Obfuscation – concealing program's logic, purpose, or behaviour.

Issues with obfuscation:

- Does not target keys specifically.
- Existing techniques are vulnerable to static and dynamic analysis.
- Theoretical results are ambiguous: generic obfuscators do not exist [Barak et al. 2001], but simple functions (e.g., point functions) can be obfuscated.
- No theoretical method that just waits for optimization.

```
typedef char z;
o A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y, E_, w[20], bf[20], bg[20],x=-1.3/8,p,d,l=0,h=4e-3,f,m;
z e[17] = "FKE<<gMAQUDIYO9\"=", *k, *j; o g(o p, o d){int q =-1,t=0;j=k;while( *j ){c(45,-);Z(100){y;q++;r=v;q[bf]=bf[q-1];q[bg]=bg[q-1];}Z(42)
    {y;v*=r;o(b,[q-1]=r*b,[q-1] + v*b,[q]);}Z(44){t=0;q++;}Z(116){f=v;v=r;r=f;f=bf[q];q[bf]=bf[q-1];bf[q-1]=f;f=q[bg];bg[q]=(q-1)[bg];bg[q-1]=f;



     }c(43,+);Z(47){y;v/=r;              o(b,[q-1]=b,[q-1]/r-v/              (r*r)*b,[q]);}Z(121){y;q++                  ;r=d; bf[ q]=0;bg[q]=1;}
     Z(94){y;f=v;v=pow(v,r)              ;o(b,[q-1]=r*b,[q-1]*               pow(f,r-1)+0*b,[q]);}w(115                  ,sin,cos);w(99,cos,sin);
     Z(120){y;q++;r=p;bf[q]              =1;bg[q]=0;}if(*j>=48               &&*j<58){q++;t=1;r=*j-48 ;                  bf[q] = 0;(q--)[bg] = 0;
                             }j++;}H=bf[0]                                               ;K= bg[ 0]; return w[0];}
```

More on obfuscation

Let us elaborate more on obfuscation:

- If we can not make provably unbreakable, we can try to make it <u>seemingly</u> unbreakable (cf. the security of AES).
- The "obfuscation path" was initially offered for the public-key cryptography, but eventually the mathematically hard problems have been chosen.

Let us elaborate more on obfuscation:

- If we can not make provably unbreakable, we can try to make it seemingly unbreakable (cf. the security of AES).
- The "obfuscation path" was initially offered for the public-key cryptography, but eventually the mathematically hard problems have been chosen.
- So how could we obfuscate keys?
- What sort of security would we want to get?

White-box cryptography

White-box implementation

WBC centers around white-box implementation:

1. Pure software implementation of a cipher (encryption or decryption routine) with embedded key;
2. Implementation is assumed available to an adversary;
3. Adversary gets little to no advantage over a black-box implementation, where only inputs and outputs are observed.

White-box implementation

WBC centers around white-box implementation:

1. Pure software implementation of a cipher (encryption or decryption routine) with embedded key;
2. Implementation is assumed available to an adversary;
3. Adversary gets little to no advantage over a black-box implementation, where only inputs and outputs are observed.

Similar to public-key cryptography (RSA). Why not using it?

WBC centers around white-box implementation:

1. Pure software implementation of a cipher (encryption or decryption routine) with embedded key;
2. Implementation is assumed available to an adversary;
3. Adversary gets little to no advantage over a black-box implementation, where only inputs and outputs are observed.

Similar to public-key cryptography (RSA). Why not using it?

- RSA-2048 encryption speed — 1000 CPU cycles per byte.
- AES-128 encryption speed — 0.7 CPU cycles per byte.

WBC centers around white-box implementation:

1. Pure software implementation of a cipher (encryption or decryption routine) with embedded key;
2. Implementation is assumed available to an adversary;
3. Adversary gets little to no advantage over a black-box implementation, where only inputs and outputs are observed.

Similar to public-key cryptography (RSA). Why not using it?

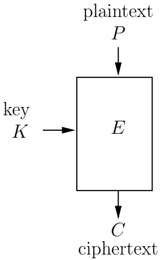
- RSA-2048 encryption speed — 1000 CPU cycles per byte.
- AES-128 encryption speed — 0.7 CPU cycles per byte.

Impractical for large amount of data. So one more requirement:

4. Performance loss should be minimal.

Apparently, our tools are limited...

Naive way to hide a key: put everything in a large lookup table.

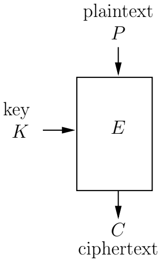


$\approx$

plaintext	ciphertext
P_1	C_1
P_2	C_2
...	...

However, conventional ciphers work with 128-bit blocks. A single table is clearly infeasible ($\approx 2^{128}$ size).

Naive way to hide a key: put everything in a large lookup table.



$\approx$

plaintext	ciphertext
P_1	C_1
P_2	C_2
...	...

However, conventional ciphers work with 128-bit blocks. A single table is clearly infeasible ($\approx 2^{128}$ size).

Smaller key-dependent tables?

- Implements any function in given domain;
- Memory-consuming (4 GB for 32-bit tables);
- Easily invertible.

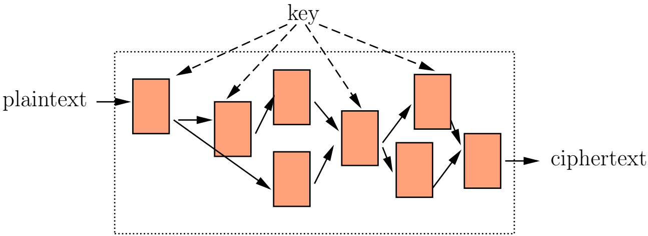
White-box implementation of AES

First proposal

Chow, Eisen, Johnson, and van Oorschot, "White-Box Cryptography and an AES Implementation" (2002).

- Obfuscate AES implementation with embedded keys;
- Publish algorithm as a sequence of smaller table lookups.

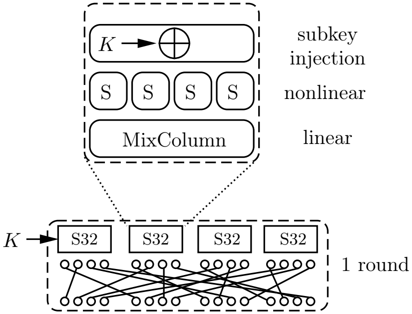
Goal: make the key recovery difficult.



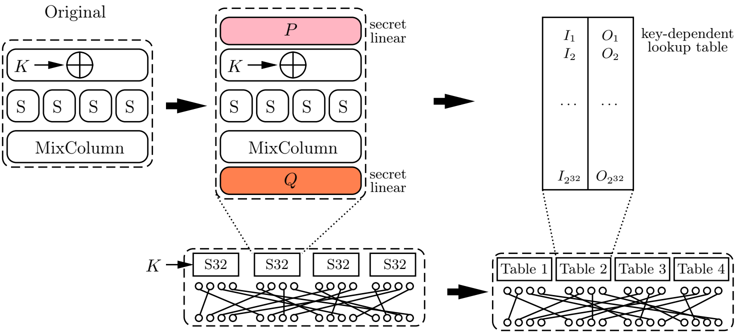
AES-128 (designed in 1997, adopted in 2001): 10-round cipher with 16-byte state.

One round of AES:

- Four 32-bit blocks:
  - AddRoundKey (simple XOR);
  - SubBytes (bytewise nonlinear);
  - MixColumns (linear).
- ShiftRows (byte permutation).

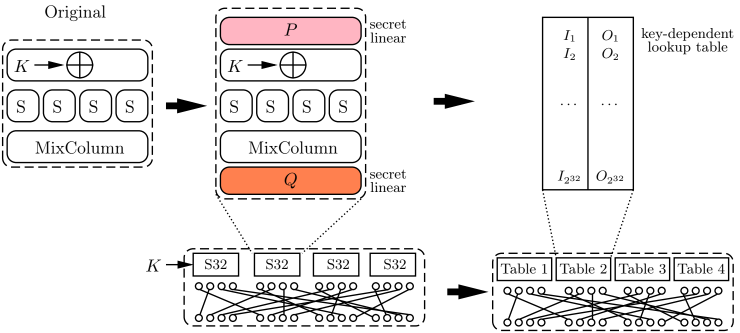


Simplistic view of white-box encoding:



- Add secret, random, mutually compensative transformations P and Q;
- Replace every 32-bit block with a lookup table;
- Store everything in memory.

Simplistic view of white-box encoding:

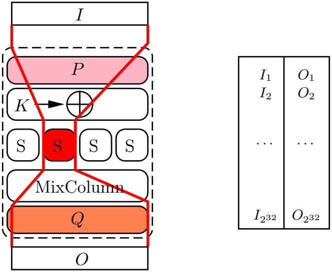


- Add secret, random, mutually compensative transformations P and Q;
- Replace every 32-bit block with a lookup table;
- Store everything in memory.

Actual proposal used smaller and weaker tables.

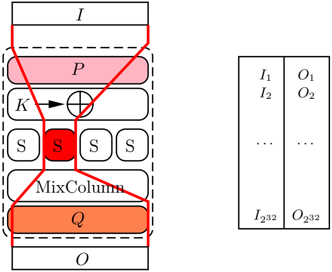
Cryptanalysis

Differential cryptanalysis applies to all proposed implementations.



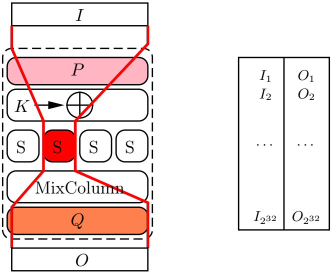
Differential cryptanalysis applies to all proposed implementations.

Consider multiple inputs $I_1, I_2, \ldots, I_k$ and the evolution of differences between them: $I_a \oplus I_b$.



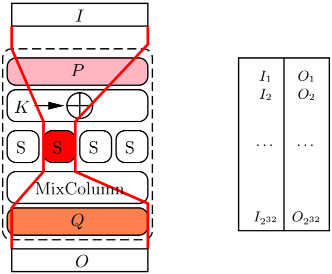
- Suppose $P(I_a \oplus I_b)$ is non-zero in only one S-box;
- Such input pairs form a linear space;
- The output differences form a matrix of low rank.

Consider multiple inputs $I_1, I_2, \ldots, I_k$ and the evolution of differences between them: $I_a \oplus I_b$.



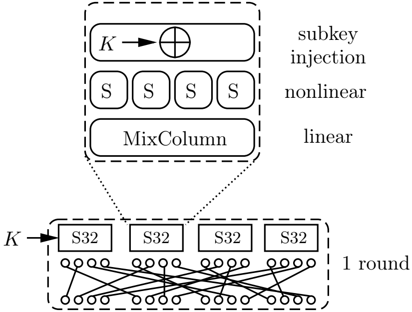
We retrieve P and Q up to linear equivalence, and then just guess the key bytewise.

With other optimizations, the entire key can be extracted from tables in 2^{30} simple operations (seconds on a PC).



Here we have only one nonlinear layer. Even 3 nonlinear layers are not enough [Biryukov-Shamir'01].

Why table-based approach fails for AES? Recall the structure:



- Small tables contain too little key material;
- Hiding entire key would require enormously large tables.

White-box implementation of AES (2002):

- Attack on the first variant [Billet'04];
- Improved variants [Bringer'06, Karroumi'11];
- Attacks on improved variants [DeMulder'10,'12,'13].

White-box implementation of DES (2002):

- Attacks on "naked variant" (fault attack [Jacob'02], statistical attack [Link'05]);
- Improved variant;
- Attacks on improved variant [Goubin'07,Wyseur'07].

All attacks have practical complexity.

So is everything broken?

So is everything broken?

Principally, yes

Proprietary solutions

Still, many proprietary solutions available at the market:

- SafeNet Sentinel;
- Irdeto Cloakware Security Kernel;
- Arxan TransformIT;
- whiteCryption MCFACT;
- Microsemi's Whiteboxcrypto.

They probably combine academic proposals with ad-hoc obfuscation techniques.

Still, many proprietary solutions available at the market:

- SafeNet Sentinel;
- Irdeto Cloakware Security Kernel;
- Arxan TransformIT;
- whiteCryption MCFACT;
- Microsemi's Whiteboxcrypto.

They probably combine academic proposals with ad-hoc obfuscation techniques.

No public attacks.

We have talked about key recovery only. What about other security goals?

What security should we expect from a white-box implementation?

Weak WBC

Key-recovery security: an adversary can not extract the key from the code.

- Chronologically first definition;
- Apparently easier to achieve;

What security should we expect from a white-box implementation?

Weak WBC

Key-recovery security: an adversary can not extract the key from the code.

- Chronologically first definition;
- Apparently easier to achieve;
- Irrelevant when the code can be extracted and isolated easily (code lifting).
- Still makes sense if code lifting is difficult (e.g., the software is watermarked and traceable).

What security should we expect from a white-box implementation?

Weak WBC

Key-recovery security: an adversary can not extract the key from the code.

- Chronologically first definition;
- Apparently easier to achieve;
- Irrelevant when the code can be extracted and isolated easily (code lifting).
- Still makes sense if code lifting is difficult (e.g., the software is watermarked and traceable).

Strong WBC

Plaintext-recovery security: an adversary can not invert the cipher, i.e. decrypt given the encryption routine.

- More sound definition;
- Applies also in the case of code lifting;
- Almost identical to public-key cryptography (may replace it).
- All existing proposals do not comply.

New approaches to WBC

White-box implementation from scratch

Problems with existing ciphers:

- Not designed with white-box implementations in mind;
- Even key-recovery security is difficult to achieve.

What if we make a white-box suitable cipher from scratch?

Let's start with key-recovery security (weak WBC).

Weak white-box proposal

It is apparently easy to hide a key in a small table.

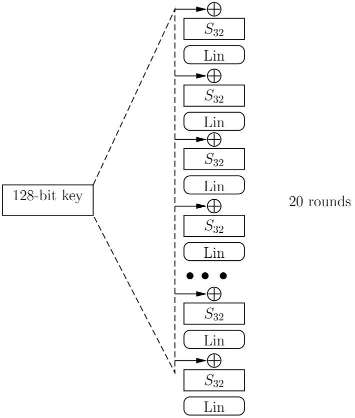
It is apparently easy to hide a key in a small table.

$32 * t$-bit block cipher with 128-bit key:

- 20 rounds;
- Round: subkey injection + 32-bit S-box (taken from AES)+ linear transformation;
- Each key byte is used 5 times.
- Make a lookup table for all 2^{32} inputs and given key.
- Mix the tables linearly if wideblock-cipher is needed.

Security margin larger than in AES...

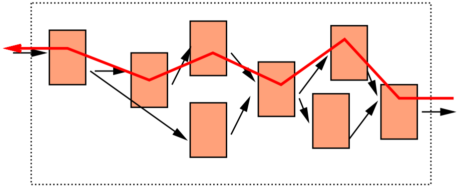
but trivially invertible.



How to make the construction non-invertible without a key?

Lookup table problems:

- Lookup tables allow inputs up to 32 bits only;
- Any network of lookup tables are usually trivially invertible (search in the table);
- We do not know how to do otherwise (open problem!).



Non-invertibility from functions

Can we use functions (one-way permutations) that are difficult to invert?

Non-invertibility from functions

Can we use functions (one-way permutations) that are difficult to invert?

- If it is easy to invert with a key, we face a trapdoor permutation.

Non-invertibility from functions

Can we use functions (one-way permutations) that are difficult to invert?

- If it is easy to invert with a key, we face a trapdoor permutation.
- Known candidates such as RSA

$$x \to x^3 \pmod{N}$$

  are believed secure for N of thousand bits long only.
- Still open problem for smaller N and reasonable performance.

How about other algebraic constructions?

We know that inverting a random degree-2 vector-polynomial is hard

$$(x_1, x_2, \ldots, x_n) \to (x_1 x_2 + x_3 x_7 + \cdots + x_8 x_n + x_{n-3}, \ldots).$$

The problem is that we can not make it random enough to hide a trapdoor there.

Public-key cryptography with polynomials:

$$\mathbf{b} = T \circ \mathbf{a} \circ S, \tag{1}$$

where S and T are key-dependent and secret, and $\mathbf{a}$ is a public invertible polynomial of degree 2.

- Degree-2 polynomials of 128 boolean variables are compact enough (less than 1 MByte), and there is no generic inversion algorithm.

Public-key cryptography with polynomials:

$$\mathbf{b} = T \circ \mathbf{a} \circ S, \tag{1}$$

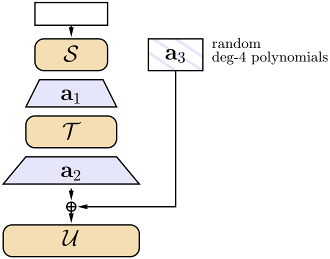
where S and T are key-dependent and secret, and $\mathbf{a}$ is a public invertible polynomial of degree 2.

- Degree-2 polynomials of 128 boolean variables are compact enough (less than 1 MByte), and there is no generic inversion algorithm.
- However, virtually all variants of this scheme have been broken because of properties of $\mathbf{a}$: only a few families of invertible polynomials are available (even without trapdoors).

Some work in progress:

$$\mathbf{b} = U \circ \mathbf{a}_2 \circ T \circ \mathbf{a}_1 \circ S$$

- Two nonlinear layers;
- Nonlinear transformations are expanding and more random-looking;
- Noise ($\mathbf{a}_3$) added to defeat generic decomposition algorithms.



Summary

Summary

- White-box cryptography aims to obfuscate encryption or decryption routines with embedded keys to make the key extraction or inversion impossible;
- It is quite similar to public-key cryptography and generic obfuscation;
- All academic proposals are weak;
- Many proprietary solutions available with unknown basis and security level;
- Good solutions may not exist.

Questions?