

Evolution of White-Box Cryptography: From Table-Based Implementations to Recent Designs

Michael J. Wiener

2016 August 14



- Why do we bother with white-box cryptography (WBC)?
- The origins of WBC
- Attacks and countermeasures
 - BGE, DFA, DCA
- New generations of WBC designs
- Research needed
 - Theory of security
 - Indistinguishability obfuscation
 - White-box friendly ciphers
- Other directions for software security
 - Trusted execution environments
 - Homomorphic encryption
- Conclusions

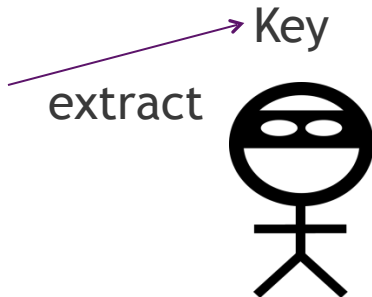
- In industry, we use a pragmatic definition of WBC:

White-box cryptography is the design of software implementations of cryptographic algorithms that resist attack.

- What is an attack? Some possibilities:
 - We know it when we see it.
 - Anything that disrupts business.
 - Anything that creates a viable business for the attackers.

Key extraction

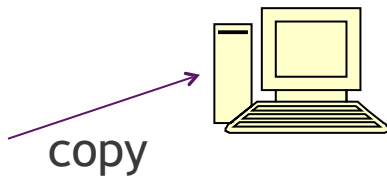
WBC implementation



- WBC tries to keep keys hidden while encrypting with them.
- Attackers try to extract these keys.

Code lifting

WBC implementation



- Attackers try to lift the entire implementation instead of just extracting a key
- We use techniques to lock software to a given platform.

- Early in its history, there was a misconception that WBC competes with traditional cryptography.

Modern cryptography:

- Tremendous advances:
 - AES
 - Public key
 - Secure protocols
- All just theoretical before we implement in hardware or software

WBC:

- Complements traditional cryptography
- In theory, AES takes trillions of years to break
- Without WBC, reading an AES key out of memory may only take seconds

- If WBC has shortcomings, we cannot just go back to traditional cryptography.
- We have to implement algorithms securely somehow.

- If WBC is important, how have we got along without it?



- In reality, WBC is in widespread use in many applications.
- Details are kept secret.

- If your smart phone only contains trusted software and is only ever under the physical control of trustworthy people, then you don't need WBC.



- However, if you live on earth where we have attackers and malware, you may benefit from using WBC.

The claim:

“WBC is less secure than traditional (black-box) cryptography.”



What they are comparing:

WBC implementations +

Attackers who have full software access

vs.

Unprotected black-box cryptographic implementations +

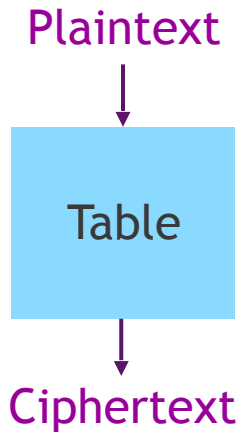
Attackers with no side-channel access or software access

We have limited control over the attacker’s powers.

The work we do on WBC is a reflection of the reality of attacker powers. We cannot wish away software vulnerability.

Pioneering work on WBC was done at Cloakware (later acquired by Irdeto) by Chow et al. for AES [CEJvO] and DES [CEJvO2].

This work began with a simple observation:



For a given encryption key, building a table mapping all input data (plaintext) to output data (ciphertext) is

- Hopelessly impractical (for AES, 2^{128} entries)
- But secure against key extraction!

This led to the idea of using many small tables instead of one big one.

For a complete tutorial on the first AES design, see James Muir's paper [M13].

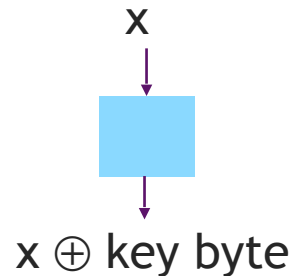
We start with an insecure table-based AES implementation and look at how Chow et al. modified it.

3 of the 4 AES steps are quite straightforward.

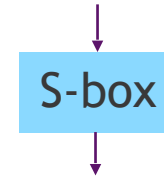
ShiftRows

Just moves bytes around. No tables needed.

AddRoundKey



SubBytes



MixColumns involves expanding 8 bits to 32 bits and XORing four 32-bit values.

- There are 4 different expansions (Ty0, Ty1, Ty2, Ty3).

XORing is done 4 bits at a time to control table size.

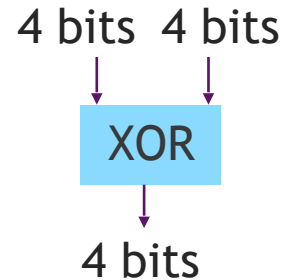
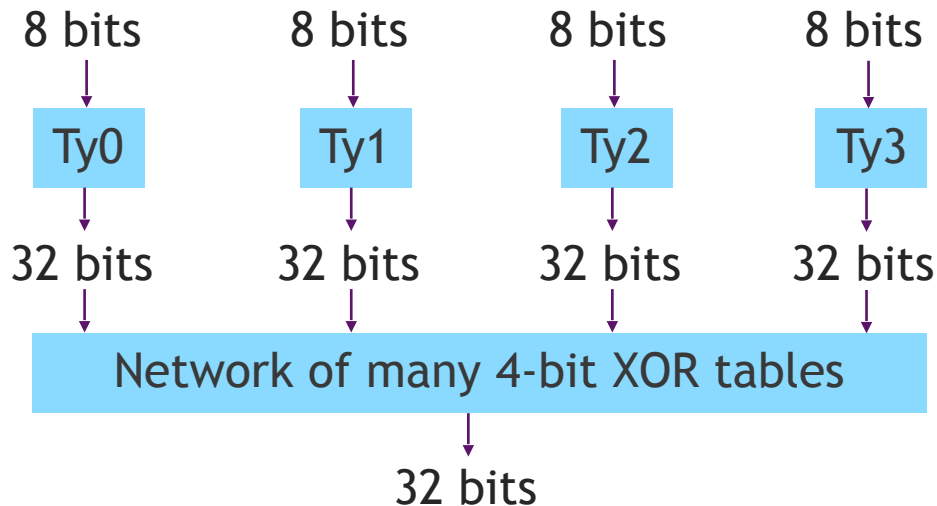
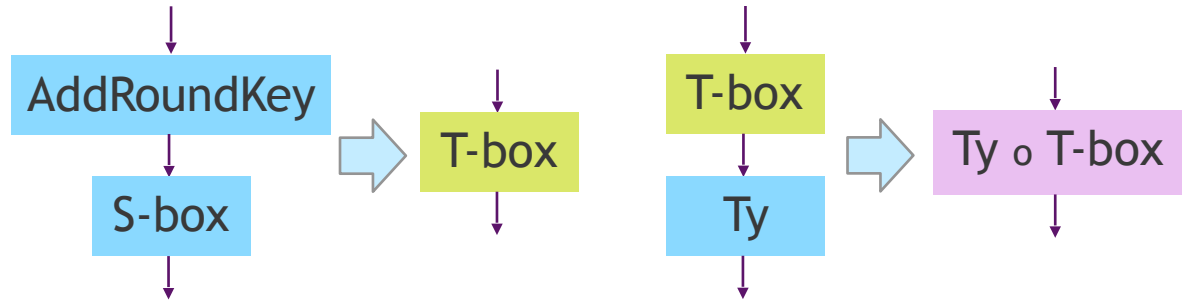


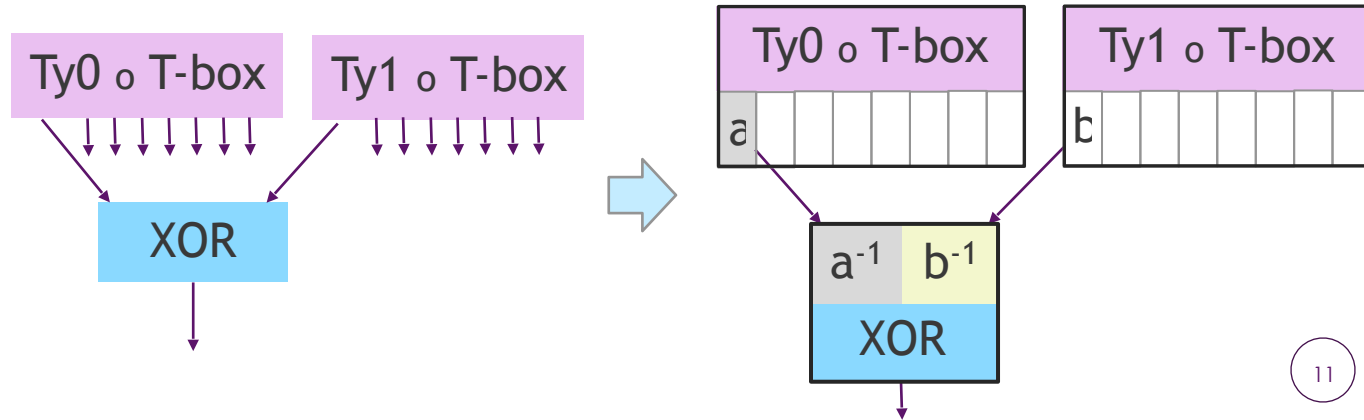
Table Composition

Whenever one table feeds directly into another table, replace the two tables with one composed table.



Random Bijections

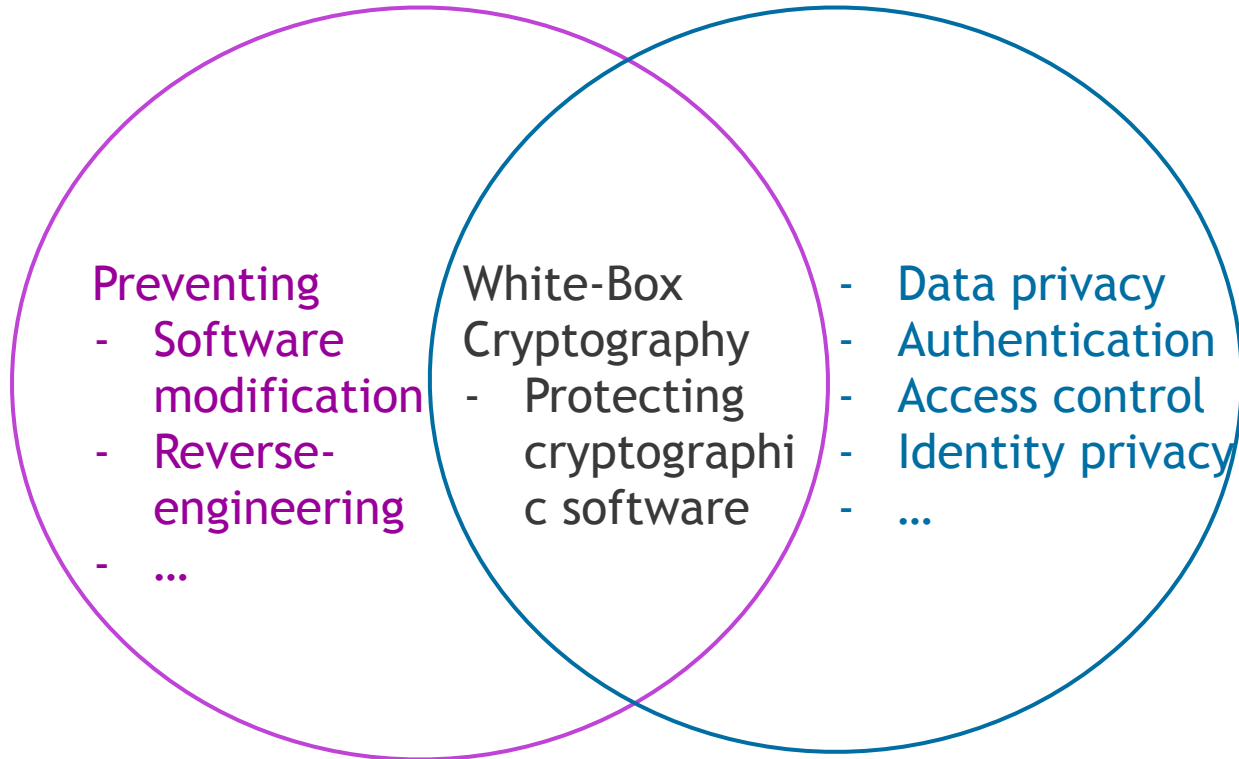
Compose concatenated 4-bit random bijections (a and b) that cancel.



- Many people consider WBC to mean table-based designs similar to the original Chow et al. AES and DES implementations.
 - We take a much broader view.
- WBC is any attempt to design attack-resistant cryptographic software. E.g.,
 - Different types of tables
 - Possibly no tables at all
 - Protections using software security methods
 - ...

Software Protection

Cryptography




Security is very important, but hard to measure.

Security tends to get focus after a successful attack in the field.

Until then, size and speed concerns dominate.

WBC software vs. unprotected cryptographic software:

10x bigger and slower  Usually acceptable

100x bigger and slower  Sometimes acceptable

1000x bigger and slower  Almost always unacceptable

All security measures must give enough benefit to justify their costs.

- Chow et al. made important contributions.
 - But they gave us just the 1st generation of WBC.
 - This is good because this 1st generation has been thoroughly broken.
- Billet et al. found the first attack [BGE] on the published white-box AES [CEJvO]. They were able to extract the AES key.

We investigated each white-box table further:

Each table
comprises multiple
operations

Table
 $= f \circ g \circ h$



Extract
f, g, h

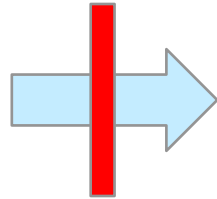
We were able to extract
all table components up to
the information theoretic
bound.

The break was thorough and we later developed countermeasures for new white-box implementations.

The details of this 2nd generation of white-box AES and triple-DES were never published.

Barak et al. proved:

There exist programs that cannot be protected [BGIRSVY].



Some people conclude:

WBC cannot work.
Not true.

In the practical world we do not need to protect all programs.

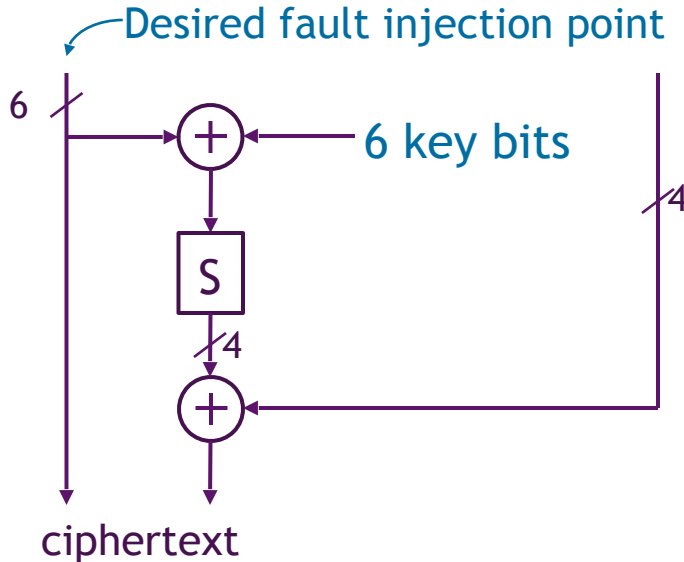
It is clearly possible to make the life of hackers and reverse-engineers harder.

- The second generation of WBC designs resisted attacks like that of Billet et al. [BGE], but fell to two new attacks borrowed from hardware side channel work.
 - Differential Fault Analysis (DFA)
 - Differential Computation Analysis (DCA)
- We needed to move on to a third generation of WBC designs.

DFA was originally an attack on hardware crypto implementations [BS97].

But it works on white-box implementations as well.

An example showing part of the final stage of DES:



Attacker first runs code and records ciphertext.

Then injects fault and records faulty ciphertext.

Only certain combinations of 6 key bits are consistent with the results.

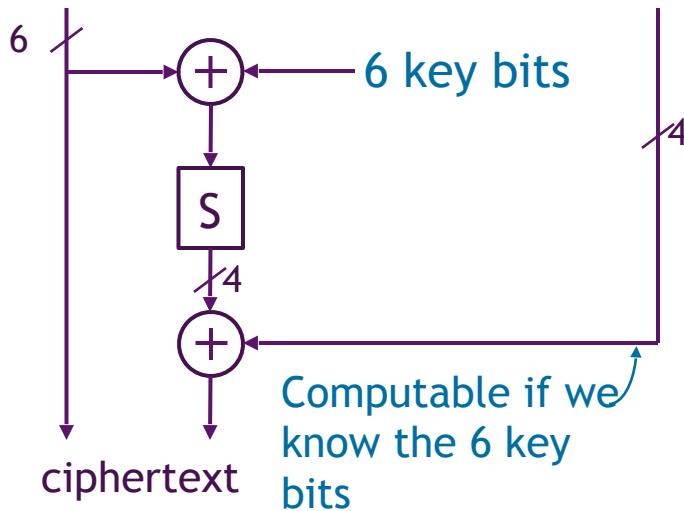
Rinse and repeat until entire key is known.

White-Box DFA Attack Requirements	Countermeasures
Ability to inject the right types of faults	<ul style="list-style-type: none"> • Redundant computations to detect faulty data • Disguise redundancy with distinct data transforms • Corrupt ciphertext if redundant computations do not match
Ability to make faults that affect just a few bits	<ul style="list-style-type: none"> • Transformations on data with wide scope • Changing transformed data affects many 'original' bits
Ability to view ciphertext	<ul style="list-style-type: none"> • Apply data transformations to ciphertext
<p>This led to a 3rd generation of white box AES and triple-DES whose details were never published.</p>	

DCA [BHMT] is based on Differential Power Analysis (DPA) [KJJ99].

DPA extracts keys from physical crypto implementations by examining power usage.

An example (showing part of the final stage of DES):



Choose a guess of the 6 key bits.

Execute DES for many random plaintexts, record ciphertext and power consumption traces.

Compute one internal bit for each trace.

Average the 0-traces and 1-traces separately.

If there is a non-random difference between the averages, then the 6-bit key

DPA uses power traces. DCA uses computation traces.

Computation traces can be

- Snapshots of memory during white-box execution
- Stack contents
- Table indexes
- Register contents
- Pointer values

With DCA, we look for correlations between the internal computed bit and any part of the computation traces. Finding a correlation means the key bits guess is right.

This attack is powerful because the attacker does not need to know in advance which part of the computation traces will reveal a correlation.

White-Box DCA Attack Requirements	Countermeasures
Existence of correlations between white-box data representations and the untransformed data	<ul style="list-style-type: none">• Ensure no such correlations exist• Requires a careful entangling of intermediate data throughout the software
Ability to view ciphertext	<ul style="list-style-type: none">• Apply data transformations to ciphertext

Some countermeasures in our 3rd generation white-box implementations resisted DCA.

However, we have a 4th generation with improved resistance.

An important part of a security strategy is continuous adaptation to limit the damage from attacks.

- We have had success creating attack-resistant white-box software.
 - But we lack a theory that allows us to link WBC security to a known hard problem.
 - Perhaps indistinguishability obfuscation (iO) will be a useful research path.
- A big part of the challenge is the need for size and speed efficiency.
- We welcome new research advances in WBC, but do not have the luxury of time to wait for them.
 - The need for secure software today is great.

So far we have focused on AES and DES.

But there are many other cryptographic algorithms:

- RSA
- Elliptic Curves
- Hash functions
- Message authentication codes
- ...

The rich mathematical structure of most public-key algorithms makes them particularly challenging to protect in software.

I will not say more here about these algorithms other than to say that each presents unique white-box challenges and we have designs for each.

- Those familiar with the original WBC papers tend to think in terms of “fixed-key” software designs
 - Fixed-key means the cryptographic key is baked into the software.
- However, in many applications, we need “variable-key” designs
 - Variable-key means the software takes a protected key as an input so that it can operate with different keys at different times.
- We have fixed-key and variable-key versions of our WBC designs.

The world knows white-box cryptography as table-based implementations similar to the first published papers.

We are examining ways to retain the mathematical complexity of these table-based designs, but without any tables - just code.

Tables	Code
Tables stand out in software.	White-box code blends better with other parts of an application. It is better if attackers cannot see where white-box crypto begins and ends.
We seem to be at the limit for the types of operations we can compose with tables.	We can better leverage growing research into software protection.
An interesting side effect of table-free designs is that it has made it easier to adapt to protecting new cryptosystems.	

- Coming up with AES was a world-wide effort.
 - Designing your own cipher makes little sense from a black-box point of view.
- But perhaps there are ciphers that are easier to protect in a white-box environment than AES is.
 - “White-box friendly” ciphers
- Another potential advantage of custom ciphers is that it is hard to mount DFA or DCA attacks against unknown ciphers.
 - Frequent updates of the cipher could be a useful security measure.
- This is an active area of research.

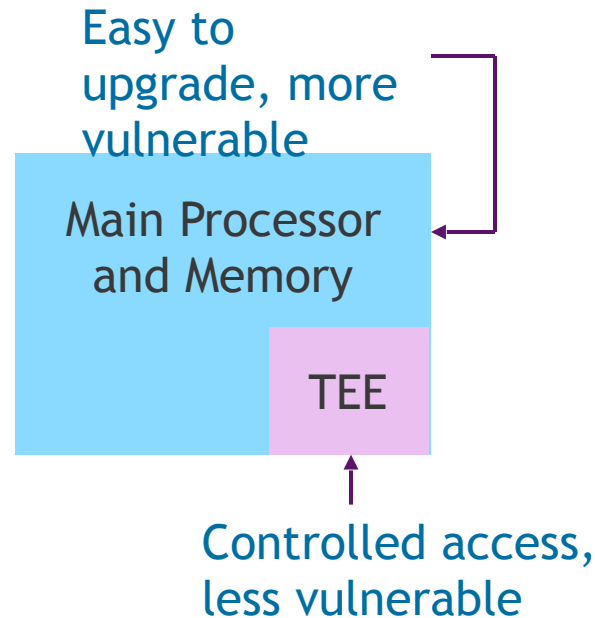
One of the available tools for improving software security is a Trusted Execution Environment (TEE).

TEEs use hardware to limit access to programs and data during execution.

TEEs are an important part of software security but are not perfect; we should still protect the software they run.

For security, we limit TEE access to one or more trusted software vendors.

Relentless pressure for openness leads to opening TEE access and reducing its security over time.



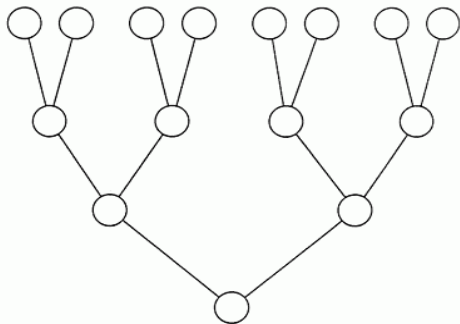
Homomorphic encryption (HE) is a great achievement of modern cryptography.

It allows us to compute with encrypted data without decrypting the data.

HE seems like it should be an important part of software security.

Setting aside inefficiency, HE has some limitations.

Acting on Decisions



0 or 1,
but which is it?

We can compute a decision:

0 = customer did not approve
transaction

1 = customer approved transaction

But without the key, we cannot read
the bit to decide whether to send a
payment.

If the vulnerable software has the key,
this defeats the purpose of HE.

Exposing computations

HE hides data effectively, but it does not naturally hide the computations being performed.

An attacker can see what computation is being performed and potentially change it to a new computation.

It would be possible to get around this problem by creating an HE virtual machine where instructions are encrypted. But this would add another layer of inefficiency.

In our software-dominated world, the need for software protection is unavoidable.

Encryption software is often the most security-critical part of an application, and must be protected with some form of WBC.

We have had success preventing attacks such as DFA and DCA with our latest generations of WBC designs, but research is needed.

Some pin their hopes on trusted execution environments and homomorphic encryption, but they are unlikely to be complete solutions.

- [BGE] O. Billet, H. Gilbert, and C. Ech-Chatbi, “Cryptanalysis of a White Box Implementation,” pp. 227-240, Selected Areas in Cryptography 2004, Lecture Notes in Computer Science Vol. 3357, H. Handschuh and A. Hasan eds., Springer 2005.
- [BGIRSVY] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. “On the (Im)possibility of Obfuscating Programs (Extended Abstract),” pp. 1-18, Advances in Cryptology – Crypto 2001, Lecture Notes in Computer Science Vol. 2139, J. Kilian ed., Springer 2001.
- [BHMT] J. Bos, C. Hubain, W. Michiels, and P. Teuwen, “Differential Computation Analysis: Hiding Your White-Box Design is Not Enough,” Cryptology ePrint Archive, Report 2015/753, <https://eprint.iacr.org/2015/753>, 2015.
- [BS97] E. Biham and A. Shamir, “Differential fault analysis of secret key cryptosystems,” pp. 513-525, Advances in Cryptology – Crypto ’97 Proceedings, Lecture Notes in Computer Science Vol. 1294, , B. Kaliski, ed., Springer 1997.
- [CEJvO] S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot, “White-Box Cryptography and an AES Implementation,” pp. 250-270, Selected Areas in Cryptography 2002, Lecture Notes in Computer Science Vol. 2595, K. Nyberg and H. Heys eds., Springer 2003.
- [CEJvO2] S. Chow, P. Eisen, H. Johnson, and P.C. van Oorschot, “A White-Box Cryptography DES Implementation for DRM Applications,” pp. 1-15, ACM Workshop on Digital Rights Management 2002, Lecture Notes in Computer Science Vol. 2696, J. Feigenbaum ed., Springer 2003.
- [K96] P. Kocher, “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” pp. 104-113, Advances in Cryptology – Crypto ’96 Proceedings, Lecture Notes in Computer Science Vol. 1109, N. Kobitz ed., Springer 1996.
- [KJJ99] P. Kocher, J Jaffe, and B. Jun, “Differential Power Analysis,” pp. 388-397, Advances in Cryptology – Crypto ’99 Proceedings, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer 1999.
- [M13] J. A. Muir, “A Tutorial on White-Box AES,” pp. 209-229, Advances in Network Analysis and its Applications, Mathematics in Industry 18, 2013.
- [PQ03] G. Piret and J.-J. Quisquater, “A differential fault attack technique against SPN structure, with application to the AES and KHAZAD,” pp. 77-88, Cryptographic Hardware and Embedded Systems – CHES 2003, Lecture Notes in Computer Science Vol. 2779, C. Walter, C. Koc, and C. Paar eds., Springer 2003.
- [W15] M. Wiener, “Applying Software Protection to White-Box Cryptography,” Proceedings of the 5th Program Protection and Reverse Engineering Workshop, Los Angeles/Universal City, CA, 2015.