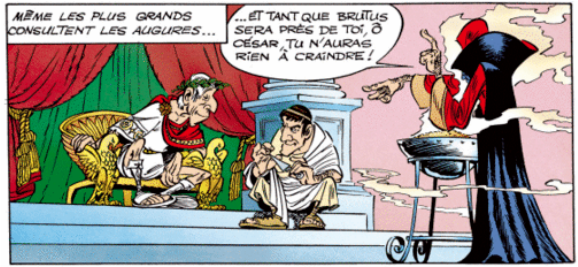
CAESAR Implementations & BRUTUS Plans for Protocols

Markku-Juhani O. Saarinen
mjos@iki.fi

https://mjos.fi

FSE '15 Rump Session – 10 March 2015 – Istanbul, TURKEY

Background: BRUTUS and CAESAR

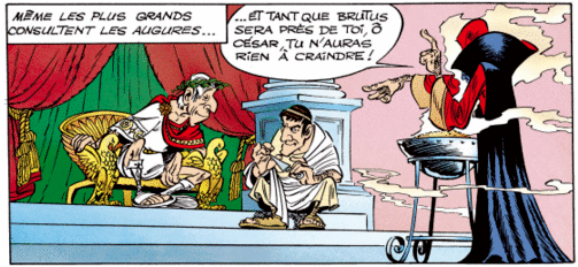


BRUTUS is a testing framework for **50+ CAESAR** ciphers. Implementations are compiled as dynamically loadable modules; this allows more rapid experimentation when compared to **SUPERCOP** (which was intended only for speed tests anyway).

Markku-Juhani O. Saarinen: *"BRUTUS: Identifying Cryptanalytic Weaknesses in CAESAR First Round Candidates."* Available at: `http://eprint.iacr.org/2014/850`

BRUTUS source code available at: `https://github.com/mjosaarinen/brutus/`

Background: BRUTUS and CAESAR



BRUTUS is a testing framework for **50+ CAESAR** ciphers. Implementations are compiled as dynamically loadable modules; this allows more rapid experimentation when compared to **SUPERCOP** (which was intended only for speed tests anyway).

Markku-Juhani O. Saarinen: *"BRUTUS: Identifying Cryptanalytic Weaknesses in CAESAR First Round Candidates."* Available at: `http://eprint.iacr.org/2014/850`

BRUTUS source code available at: `https://github.com/mjosaarinen/brutus/`

Protip: C does not have garbage collection

Authors of MARBLE, CLOC, SILC, LAC, and POET reference implementations:

C is not Java. You need to pick up your litter.

These implementations have a bunch of `malloc()` calls but zero `free()` calls.

- This **will**, in the long run, crash any application using these implementations.
- Memory leakage has far more serious security implications than any cryptanalytic weakness that MARBLE, CLOC, SILC, LAC, or POET may have.
- All reasonable implementations of on-line ciphers avoid dynamic memory allocation altogether since it should not be necessary (think embedded).

Protip: C does not have garbage collection

Authors of MARBLE, CLOC, SILC, LAC, and POET reference implementations:

C is not Java. You need to pick up your litter.

These implementations have a bunch of `malloc()` calls but zero `free()` calls.

- This **will**, in the long run, crash any application using these implementations.
- Memory leakage has far more serious security implications than any cryptanalytic weakness that MARBLE, CLOC, SILC, LAC, or POET may have.
- All reasonable implementations of on-line ciphers avoid dynamic memory allocation altogether since it should not be necessary (think embedded).

You don’t care much about security engineering, but..

What is ..

“ `byte *res = (byte *)malloc(5+nbytes+adbytes+padbytes);`”

.. in Verilog or VHDL ?

There is no such thing as dynamic memory in hardware. What’s the HW API?

You don’t care much about security engineering, but..

What is ..

“ `byte *res = (byte *)malloc(5+nbytes+adbytes+padbytes);`”

.. in Verilog or VHDL ?

There is no such thing as dynamic memory in hardware. What’s the HW API?

Additional notes for Reference Implementations (quickly)

- There are little-endian and big-endian computers and your reference code should give the same results on both (many submissions).
- There are alignment limitations on many platforms – some systems will halt if you read (big) words from unaligned addresses (many submissions).
- C source code files have `.c` suffix and C++ source files have `.cpp` suffix. If you put C functions into a `.cpp` file, linkage will be incompatible (PAEQ, Primates).
- In C, source code and data of functions go into `.c` files and prototypes and definitions go into `.h` files (SABLIER, ELMD, AES-OTR, SHELL).

Implementations have to be heavily modified for real life usage w. **context structures**, **constant-time operation**, **clearup of sensitive data**, etc.. hence:

With universal reference implementations, please sacrifice your *perceived* performance optimizations for uniform, correct operation on *all* platforms.

Additional notes for Reference Implementations (quickly)

- There are little-endian and big-endian computers and your reference code should give the same results on both (many submissions).
- There are alignment limitations on many platforms – some systems will halt if you read (big) words from unaligned addresses (many submissions).
- C source code files have `.c` suffix and C++ source files have `.cpp` suffix. If you put C functions into a `.cpp` file, linkage will be incompatible (PAEQ, Primates).
- In C, source code and data of functions go into `.c` files and prototypes and definitions go into `.h` files (SABLIER, ELMD, AES-OTR, SHELL).

Implementations have to be heavily modified for real life usage w. **context structures**, **constant-time operation**, **clearup of sensitive data**, etc.. hence:

With universal reference implementations, please sacrifice your *perceived* performance optimizations for uniform, correct operation on *all* platforms.

Additional notes for Reference Implementations (quickly)

- There are little-endian and big-endian computers and your reference code should give the same results on both (many submissions).
- There are alignment limitations on many platforms – some systems will halt if you read (big) words from unaligned addresses (many submissions).
- C source code files have `.c` suffix and C++ source files have `.cpp` suffix. If you put C functions into a `.cpp` file, linkage will be incompatible (PAEQ, Primates).
- In C, source code and data of functions go into `.c` files and prototypes and definitions go into `.h` files (SABLIER, ELMD, AES-OTR, SHELL).

Implementations have to be heavily modified for real life usage w. **context structures**, **constant-time operation**, **clearup of sensitive data**, etc.. hence:

With universal reference implementations, please sacrifice your *perceived* performance optimizations for uniform, correct operation on *all* platforms.

Then there were the total disaster implementations..

"So, avalanche, huh?"

In addition to the mode of operation being equivalent to ECB (*whoops*), I was impressed by the $O(n^2)$ associated data authentication algorithm. This means that processing, say, a 64kB message takes $64^2 = 4096$ times longer than a 1 kB message.

Work plan for BRUTUSr2 and further CAESAR Experimentation

- Will update BRUTUS with R2 Tweaks.
- Automated KAT validation (now manual).
- Hardware API integration (already via SÆHI).
- OpenSSL / LibreSSL / BoringSSL / "JulianSSL":
  - Use the "engine" plugin mechanism for OS integration. Experiment with protocols.
  - This will yield realistic performance profiles.
  - Ultimately integration profiles for **TLS**, **IPSec**, **SSH2** protocols as IETF Internet-Drafts.
- This will be helpful in CAESAR adoption, perhaps replacing legacy ciphers and AES-GCM by 2020s. *(Or.. If we involve IETF CFRG, perhaps 2030s..)*

`https://github.com/mjosaarinen/brutus/`

Work plan for BRUTUSr2 and further CAESAR Experimentation

- Will update BRUTUS with R2 Tweaks.
- Automated KAT validation (now manual).
- Hardware API integration (already via SÆHI).
- OpenSSL / LibreSSL / BoringSSL / "JulianSSL":
  - Use the "engine" plugin mechanism for OS integration. Experiment with protocols.
  - This will yield realistic performance profiles.
  - Ultimately integration profiles for **TLS**, **IPSec**, **SSH2** protocols as IETF Internet-Drafts.
- This will be helpful in CAESAR adoption, perhaps replacing legacy ciphers and AES-GCM by 2020s. *(Or.. If we involve IETF CFRG, perhaps 2030s..)*

`https://github.com/mjosaarinen/brutus/`

Work plan for BRUTUSr2 and further CAESAR Experimentation

- Will update BRUTUS with R2 Tweaks.
- Automated KAT validation (now manual).
- Hardware API integration (already via SÆHI).
- OpenSSL / LibreSSL / BoringSSL / "JulianSSL":
  - Use the "engine" plugin mechanism for OS integration. Experiment with protocols.
  - This will yield realistic performance profiles.
  - Ultimately integration profiles for **TLS**, **IPSec**, **SSH2** protocols as IETF Internet-Drafts.
- This will be helpful in CAESAR adoption, perhaps replacing legacy ciphers and AES-GCM by 2020s. *(Or.. If we involve IETF CFRG, perhaps 2030s..)*

`https://github.com/mjosaarinen/brutus/`

Work plan for BRUTUSr2 and further CAESAR Experimentation

- Will update BRUTUS with R2 Tweaks.
- Automated KAT validation (now manual).
- Hardware API integration (already via SÆHI).
- OpenSSL / LibreSSL / BoringSSL / "JulianSSL":
  - Use the "engine" plugin mechanism for OS integration. Experiment with protocols.
  - This will yield realistic performance profiles.
  - Ultimately integration profiles for **TLS**, **IPSec**, **SSH2** protocols as IETF Internet-Drafts.
- This will be helpful in CAESAR adoption, perhaps replacing legacy ciphers and AES-GCM by 2020s. *(Or.. If we involve IETF CFRG, perhaps 2030s..)*

`https://github.com/mjosaarinen/brutus/`

Work plan for BRUTUSr2 and further CAESAR Experimentation

- Will update BRUTUS with R2 Tweaks.
- Automated KAT validation (now manual).
- Hardware API integration (already via SÆHI).
- OpenSSL / LibreSSL / BoringSSL / "JulianSSL":
  - Use the "engine" plugin mechanism for OS integration. Experiment with protocols.
  - This will yield realistic performance profiles.
  - Ultimately integration profiles for **TLS**, **IPSec**, **SSH2** protocols as IETF Internet-Drafts.
- This will be helpful in CAESAR adoption, perhaps replacing legacy ciphers and AES-GCM by 2020s. *(Or.. If we involve IETF CFRG, perhaps 2030s..)*

`https://github.com/mjosaarinen/brutus/`