

Join GitHub today
GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.
Sign up

Adiantum and HPolyC specification and test vectors

cryptography-algorithms

238 commits 2 branches 0 packages 0 releases 2 contributors MIT

Branch: master New pull request Find file Clone or download

Table of repository files and commits: benchmark, python, specification, test_vectors, third_party, .gitignore, CONTRIBUTING.md, LICENSE, README.md

Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/google/adiantum.gi
Open in Desktop Download ZIP

README.md

Adiantum and HPolyC

For many storage encryption applications, the ciphertext must be the same size as the plaintext; generally this matches the disk sector size of either 512 or 4096 bytes. This means that standard approaches like AES-GCM or RFC7539 cannot be applied. The standard solution is AES-XTS, but this has two disadvantages:

- If AES hardware is absent, AES is relatively slow, especially constant-time implementations
• Using XTS, a one-bit change to the plaintext means only a 16-byte change to the ciphertext, revealing more to the attacker than necessary.

Adiantum uses a fast hash (NH + Poly1305) and a fast stream cipher (XChaCha12) to build a construction which encrypts an entire sector at a time. On an ARM Cortex-A7 processor, Adiantum decrypts 4096-byte messages at 10.6 cycles per byte, over five times faster than AES-256-XTS. It is a "super pseudorandom permutation" over the whole sector, which means that any change to the plaintext of the sector results in an unrecognizably different ciphertext sector and vice versa.

Adiantum appears in IACR Transactions on Symmetric Cryptology, Volume 2018, Issue 4. We also document HPolyC, our first such proposal, which is slower on large messages but simpler and more key agile for small messages.

File layout

- specification/ : LaTeX sources for our paper presenting Adiantum
• test_vectors/other : Test vectors we use to validate our implementations of other primitives
• test_vectors/ours : Test vectors we generate, in JSON format
• python/ : Python implementation and test vector generation
• benchmark/ : software we used to generate the benchmarks in our paper
• third_party/ : derived works covered by a different license than our main MIT license

Notices

third_party/ includes derived works not covered by the MIT license; specifically software derived from the Linux kernel and licensed under GPLv2.

We include here a variety of algorithms and implementations; we make no guarantee they are suitable for production use.

This is not an officially supported Google product.