High-assurance field inversion for pairing-friendly primes

Benjamin S. Hvass, Diego F. Aranha, and Bas Spitters

Concordium blockchain research center, Aarhus University

Introduction. Bilinear pairings have become popular for deploying privacy-preserving cryptocurrencies, as they represent a fundamental building block for the zero-knowledge proofs required for security. ZCash is a clear example of this trend, where pairing-based zero-knowledge Succint Non-Interactive Arguments of Knowledge (zk-SNARKs) underlie private shielded transactions [8]. Another example of application of pairing-based protocols comes from the Chia blockchain, where Boneh-Lynn-Shacham (BLS) [10] signatures were adopted for improved smart transaction support.

There has been substantial progress in the past decade towards selecting parameters [6, 4, 5] and implementing pairing-based cryptography efficiently in software [3]. However, current record-setting implementations rely on hand-optimized architecture-specific Assembly code for the underlying field arithmetic and a great deal of manual tuning to unlock the best performance across a range of architectures. This introduces low-level code which is both hard to audit and to verify as correct, and a number of cryptographic libraries have suffered with simple bugs as a direct consequence [12].

Due to its many optimizations efficient code can be hard to verify in a *post hoc* way. Recently, an alternative path for implementing cryptographic libraries was demonstrated as viable in the Fiat-Crypto framework [12]. By combining correct-by-design optimized low-level code with automatically generated and formally verified high-level code, it became possible to develop libraries which are both efficient and formally verified. The approach was illustrated through the implementation of field arithmetic for several standardized elliptic curves using an extensible code generation framework, capable of producing code competitive in performance with popular hand-optimized multi-precision libraries [12]. The verification steps are conducted using the Coq proof assistant, a state-of-the-art theorem prover [11]. Such high assurance cryptographic implementations have recently been adopted by the industry: Google's BoringSSL on Fiat-Crypto [12], Firefox on Evercrypt [13], and the WireGuard VPN on both. A SHA-3 implementation was made using Jasmin [1], another framework for developing high-speed and high-assurance cryptographic software.

Contributions. We implement a verified and improved version of the constant-time (i.e., its runtime does not depend on input) algorithm from [9] in the Coq proof assistant and use Fiat-Crypto to generate an efficient implementation in C.

The inversion algorithm. The inversion algorithm [9] we have implemented is a constant-time variant of the Extended Euclidean Algorithm. It consists of a constant amount of iterations of a so-called *division step*. Each of these division steps consists of a conditional swap and a few arithmetical operations including a shift, a negation and an addition. There are two variants of the inversion algorithm utilizing this division step, one of which is significantly faster.

Fiat-Crypto. Fiat-Crypto is implemented and verified in the Coq proof assistant [11]. Fiat-Crypto embeds a small C-like language in the general logical framework of Coq and makes a provable connection between mathematical definitions and the generated efficient code in C or

Rust. This allows us to implement the inversion algorithm as long as we restrict ourselves to primitives contained in the specified C-like language.

Implementation. We implemented the two variants of the constant-time algorithm from [9] in Fiat and use the framework to generate a verified and constant-time field inversion. To do this, we extended the library with a few methods including shifting of large integers.

The implementation allows for inversion modulo any prime, in particular for primes used in pairing-based cryptography. We illustrate our approach with the BLS12-381 curve used in ZCash as an efficient instantiation for pairings at the 128-bit security level[1].

We have proved functional correctness of the division step in the inversion algorithm from [9]. In future work, we will prove that when iterated this step actually computes the field inverse. This, however, requires reasoning about real and 2-adic numbers, which needs several additional libraries. The loop which iterates the division step is also not generated by Fiat, since loops are not yet supported; at the moment one has to write the loop around the generated C-code oneself. We are working on automating this. The code is available at `https://github.com/bshvass/fiat-crypto` (our contribution is approx. 1500 lines of code).

Benchmarks. The generated code was integrated in the RELIC toolkit [2], a cryptographic library containing a state-of-the-art implementation of pairings. RELIC uses a combination of hand-written Assembly with higher-level C-code, and has been used by cryptocurrency and blockchain projects. For example, it has been adopted by the Chia project for BLS signatures, by the Brave browser for the Basic Attention Token (BAT) blockchain, and used for generating test vectors for the ZCash implementation of pairings in Rust.

Integrating the code with RELIC allowed convenient benchmarking to compare the efficiency of our approach with other field inversion algorithms already implemented in the library. The results are summarized in the following tables (where the second and last entries corresponds to this work):

Algorithm	Verified	Auto generated	Leaks	Cycles
Bernstein-Yang (fast) [9]	No	No	length of p	32,384
Bernstein-Yang (fast) [9]	**Yes**	**Yes**	**length of p**	**87,733**
Extended Euclidean	No	No	p and input	157,870
Fermat's Little Theorem	No	Partially	p	296,302
Bernstein-Yang [9]	No	Partially	length of p	305,924
Bernstein-Yang [9]	**Yes**	**Yes**	**length of p**	**309,150**

Table 1: Cycle counts for field inversion measured on an Intel Core i7-8650U CPU running at 1.90GHz with HyperThreading and TurboBoost disabled.

In the tests, p was always chosen to be the BLS12-381 prime, but Fiat can generate the algorithm for any desired prime. Thus, our verified and constant-time auto-generated implementation is approximately only half as fast as an insecure competitive implementation. This is reasonable for uses where security and correctness are indispensable.

Future work To fully generate the implementations, one would need to extend Fiat with loops. Furthermore, one could extend the supported language of Fiat to be able to use more efficient C primitives which would speed up all its generated implementations. To obtain a fully

[1] `https://electriccoin.co/blog/new-snark-curve/`

verified, and constant time, compilation to Assembly, we would like to use the CompCert [7] verified C-compiler, but at the moment this does not support some GCC extensions which Fiat-Crypto relies on (128-bit integer types in particular).

References

[1] José Bacelar Almeida, Cécile Baritel-Ruet, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Alley Stoughton, and Pierre-Yves Strub. Machine-checked proofs for cryptographic standards: Indifferentiability of sponge and secure high-assurance implementations of SHA-3. In *CCS*, pages 1607–1622, 2019.

[2] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[3] Diego F. Aranha, Paulo S. L. M. Barreto, Patrick Longa, and Jefferson E. Ricardini. The Realm of the Pairings. In *Selected Areas in Cryptography*, volume 8282 of *LNCS*, pages 3–25. Springer, 2013.

[4] Diego F. Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. Implementing Pairings at the 192-Bit Security Level. In *Pairing*, volume 7708 of *LNCS*, pages 177–195. Springer, 2012.

[5] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *IACR Cryptology ePrint Archive*, 2017:334, 2017.

[6] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*, volume 3897 of *LNCS*, pages 319–331. Springer, 2005.

[7] Gilles Barthe, Sandrine Blazy, Benjamin Grégoire, Rémi Hutin, Vincent Laporte, David Pichardie, and Alix Trieu. Formal verification of a constant-time preserving C compiler. *PACMPL*, 4(POPL):7:1–7:30, 2020.

[8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *S&P'14*, pages 459–474. IEEE Computer Society, 2014.

[9] Daniel J. Bernstein and Bo-Yin Yang. Fast constant-time gcd computation and modular inversion. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):340–398, May 2019.

[10] Dan Boneh, Ben Lynn, and Hovav Shacham. Short Signatures from the Weil Pairing. *J. Cryptology*, 17(4):297–319, 2004.

[11] The Coq Development Team. *The Coq Proof Assistant, version 8.10.0*. 2019.

[12] Andres Erbsen, Jade Philipoom, Jason Gross, Robert Sloan, and Adam Chlipala. Simple High-Level Code for Cryptographic Arithmetic - With Proofs, Without Compromises. In *S&P'19*. IEEE Computer Society, 2019. https://github.com/mit-plv/fiat-crypto.

[13] Jonathan Protzenko, Bryan Parno, Aymeric Fromherz, Chris Hawblitzel, Marina Polubelova, Karthikeyan Bhargavan, Benjamin Beurdouche, Joonwon Choi, Antoine Delignat-Lavaud, Cédric Fournet, et al. Evercrypt: A fast, verified, cross-platform cryptographic provider. In *S&P'19*, pages 634–653, 2019.