Модел. и анализ информ. систем. Т. **17**, № **4** (2010) 7–16

УДК 004.052.42:004.056.55

Automated Correctness Proof of Algorithm Variants in Elliptic Curve Cryptography [1]

Anikeev M. [2], Madlener F., Schlosser A., Huss S.A., Walther C.

Southern Federal University, Taganrog, Russia
Technische Universität Darmstadt, Germany

e-mail: anikeev@users.tsure.ru

received 22 August 2010

Keywords: verification, cryptography, elliptic curves

The Elliptic Curve Cryptography (ECC) is widely known as secure and reliable cryptographic scheme. In many situations the original cryptographic algorithm is modified to improve its efficiency in terms like power consumption or memory consumption which were not in the focus of the original algorithm. For all this modification it is crucial that the functionality and correctness of the original algorithm is preserved. In particular, various projective coordinate systems are applied in order to reduce the computational complexity of elliptic curve encryption by avoiding division in finite fields. This work investigates the possibilities of automated proofs on the correctness of different algorithmic variants. We introduce the theorems which are required to prove the correctness of a modified algorithm variant and the lemmas and definitions which are necessary to prove these goals. The correctness proof of the projective coordinate system transformation has practically been performed with the help of the an interactive formal verification system √eriFun.

1. Introduction

The most common public key schemes today are based on RSA [9] and on the more recent Elliptic Curve Cryptography (ECC) [5, 7]. Although ECC offers shorter bit-lengths and, thus,faster calculations [6], RSA will stay with us for the foreseeable future for legacy reasons. When it comes to real implementations of such encryption algorithms then the question of efficiency in terms of execution time, memory usage, or power consumption of the dedicated hardware/software systems is of utmost interest. Therefore, a considerable research effort has been dedicated in the past in order to develop variants of encryption

[1] Статья печатается в авторской редакции.

[2] This research is supported by German Academic Exchange Service (DAAD) and the Ministry of Education and Science of Russian Federation in the framework of the joint program "Michail Lomonosov".

algorithms, which are more efficient compared to their basic versions. The intended cryptographic functionality of such algorithm variants has to be guaranteed to remain unchanged, however.

This paper is aimed to demonstrate how a formal verification system may be exploited to provide highly automated proofs on the correctness of algorithmic variants of ECC schemes.

2. Elliptic curve cryptography

In general case an elliptic curve over a finite field is defined as the following cubic equation:

$$E:\quad y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{1}$$

where a_1, a_2, a_3, a_4, a_6 are constants and x, y are elements of an arbitrary finite field. The set of solutions $\{(x, y)\}$, which meet the elliptic curve equation, define the points of the elliptic curve E. By defining an appropriate addition (*EC-Add*) operation and an extra point $\emptyset$, called the point at infinity, these points become an additive Abelian group with $\emptyset$ as a neutral element.

Standard RSA algorithm relies on irreversibility of exponentiation in finite fields; ECC in turn exploits the irreversibility of multiplication between a pre-selected EC point P and an arbitrary scalar k. This $k{\cdot}P$ operation can be performed by repeated *EC-Add* and *EC-Double* operations, where *EC-Double* stands for point doubling, i.e. adding an EC point to itself. The EC operations in turn are composed of basic operations in the underlying finite field.

$$\underbrace{P + P + \ \ldots \ + P + P}_{k\ times} = k{\cdot}P = R$$

with $k \in \mathbb{N}$ and $P, R \in E$.

Consider the following particular case of an elliptic curve:

$$E/K : y^2 = x^3 + ax + b, char(K) \neq 2, 3, \tag{2}$$

where K is a $GF(p)$, $a \in K$, $b \in K$, and the discriminant $\Delta = -16(4a^3 + 27b^2)$ meets the condition $\Delta \not\equiv 0 \pmod{p}$.

Let $P = (x_1, y_1) \in E(K)$ and $Q = (x_2, y_2) \in E(K)$, where $P \neq \pm Q$, $P \neq \emptyset$, $Q \neq \emptyset$. Then the *EC-Add* operation is defined as follows: $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_2 - x_1, \tag{3}$$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1. \tag{4}$$

Equations (3) and (4) represent the general case of the *EC-Add* algorithm in affine coordinate system $\mathfrak{A}$; the general case of *EC-Double* is defined as $2P = P + P = (x_3, y_3)$ where

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1, \tag{5}$$

$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1. \tag{6}$$

Both *EC-Add* and *EC-Double* algorithms contain operations of division in finite fields, which demand calculation of a multiplicative inverse — an extremely expensive operation in terms of computation time and resource usage. One of the goals in the research of efficient hardware ECC implementations is to reduce such operations by replacing them with more efficient alternatives.

The following *EC-Double* algorithm is proposed in [3] for the elliptic curve defined by (2) with $a = -3$ and Jacobian coordinate system $\mathfrak{P}$, where the projective point $(X : Y : Z)$ corresponds to the affine point $(X/Z^2, Y/Z^3)$

Algorithm 1 $\mathit{Double}_{\mathit{proj}}$

```
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $2P = (X_3 : Y_3 : Z_3)$
if $P = \emptyset$ then
    return $\emptyset$
else
    $A \Leftarrow 4X_1Y_1^2$
    $B \Leftarrow 8Y_1^4$
    $C \Leftarrow 3(X_1 - Z_1^2)(X_1 + Z_1^2)$
    $D \Leftarrow -2A + C^2$
    $X_3 \Leftarrow D$
    $Y_3 \Leftarrow C(A - D) - B$
    $Z_3 \Leftarrow 2Y_1Z_1$
    return $(X_3 : Y_3 : Z_3)$
end if
```

In order to provide compatibility with other implementations it is essential that Algorithm 1 always returns exactly the same result as *EC-Double* for all the possible input combinations. The following theorem formalizes this idea.

Theorem 1. *Let $\pi : \mathfrak{A} \to \mathfrak{P}$ and $\alpha : \mathfrak{P} \to \mathfrak{A}$ be two transformation functions to transform points to projective and affine coordinates, respectively. Given the procedures $\mathit{Double}_{\mathit{aff}}$ and $\mathit{Double}_{\mathit{proj}}$ for doubling point in affine and projective coordinates, respectively, for all $p \in \mathfrak{A}$ and all $P \in \mathfrak{P}$ holds that*

$$\mathit{Double}_{\mathit{aff}}(p) = \alpha(\mathit{Double}_{\mathit{proj}}(\pi(p))), \tag{7}$$

$$\mathit{Double}_{\mathit{aff}}(\alpha(P)) = \alpha(\mathit{Double}_{\mathit{proj}}(P)). \tag{8}$$

3. √eriFun

The √eriFun system [13] is an interactive system for the verification of statements about programs written in the functional first-order programming language $\mathcal{L}$ [12]. This language consists of definition principles for freely generated polymorphic data types, for procedures operating on these data types based on recursion, case analyses, let-expressions and functional composition, and statements (called "lemmas" in $\mathcal{L}$) about the data types and the procedures. Procedures are evaluated in a call-by-value discipline. The data types *bool* with constructors *true* and *false*, and $\mathbb{N}$ for natural numbers with constructors 0 and $^{+}(\ldots)$ for the successor function are predefined in $\mathcal{L}$. Lemmas are defined by universal quantifications using case analyses and the truth values to represent connectives. Upon definition of a data type, each argument position of a constructor is provided with a selector function, e.g. $^{-}(\ldots)$ is the selector of constructor $^{+}(\ldots)$ thus representing the predecessor function, and *hd* and *tl* are the selectors of the *list*-constructor ::.

ECs are defined over arbitrary finite fields, a structure which can be modeled by a set of axioms in pure first order logic. Hence, verification of properties of operations defined on ECs is possible in first order logic—in principle. But due to the large number of axioms and defining formulas for the operations it is not feasible to use a fully automated first order theorem prover like Vampire [8] or E [10]. Given a set of clauses describing the problem these provers return *true*, *false*, or *out-of-memory*. The latter is the most likely result when the number of clauses grows too big. Using an interactive prover like √eriFun, the user is provided with intermediate results which he can analyze and use to push the proof further in the right direction. Possible interactions are the invention of new lemmas to support the proof or interactive proof steps in the current proof goal. Thus, our approach is to model the arithmetic of $GF(p)$ finite fields with recursively defined procedures and data structures and use these procedures as a base for the implementation of point doubling and addition in elliptic curves. In a next step we proved several lemmas over the base procedures (by induction) and used them to perform an interactive *first order* proof of Theorem 1.

4. Verification

An empty project in √eriFun contains nothing but several basic definitions, namely two data types ("*bool*" and "$\mathbb{N}$"), one infix function "$>$", and fourteen elementary lemmas (e.g. transitivity of "$>$" and "$=$", irreflexivity of "$>$", etc.), which are considered valid without any proofs. In order to define and prove lemmas corresponding to Theorem 1 we have added facts describing various subject areas. First of all, arithmetic operations for natural numbers and their basic properties were introduced. Then we derived similar operations of modular arithmetic for finite fields and prove their corresponding properties. And the final challenge is introduction of data types, functions, and lemmas directly related to the chosen elliptic curve equation.

The following definition of addition of x and y modulo m is given as an example of a modular arithmetic function implemented in √eriFun.

```
**function** $addm(x : \mathbb{N}, y : \mathbb{N}, m : \mathbb{N}) : \mathbb{N} <=$
if $?0(x)$
  then $y\ mod\ m$
  else $^{+}(addm(^{-}(x), y, m))\ mod\ m$
end
```

An important step of transition from natural number arithmetics to finite fields was an introduction of Fermat's little theorem [3]. Without it, most of the facts related to division in finite fields cannot be proved.

Theorem 2 (Fermat's little theorem)**.** *If* $\gcd(a, p) = 1$*, then* $a^{p-1} \equiv 1 \pmod{p}$ *[3].*

As soon as we consider only $GF(p)$ (prime finite field) operations, the precondition of the Fermat's little theorem can be simplified to $a \not\equiv 0 \pmod{p}$, which is indeed identical to $\gcd(a, p) = 1$ if p is prime.

This theorem is defined in √eriFun 's functional language as follows.

```
**lemma** $Fermat\ little <= \forall x, p : \mathbb{N}$
  if {
    if {$p > 2$,
      if {$\mathbb{P}(p), \neg\ ?0(x\ mod\ p), false$},
      $false$},
    $powm(x, ^{-}(p), p) = 1$,
    $true$}
```

Here the symbol $\mathbb{P}$ denotes a recursively-defined function, which returns *true* if its argument is a prime number and $false$ otherwise. Its definition and correctness proof is provided in [14].

Original automated verification of Fermat's little theorem, which suits √eriFun well, has been proposed by R. S. Boyer and J. S. Moore in 1984 in their effort to prove correctness of the RSA algorithm [1]. The general idea of the proof is based on the introduction of the sequence $S(n, M, p) = [nM\ mod\ P, (n-1)M\ mod\ P, ..., 1M\ mod\ P]$. It is proved that the product of all elements of $S(p-1, M, p)$ is $(p-1)!M^{p-1}\ mod\ P$, and on the other hand $S(p-1, M, p)$ is always a permutation of $[p-1, ..., 2, 1]$ sequence, whose product of elements is obviously $(p-1)!\ mod\ P$. These facts lead to the conclusion that $M^{p-1} \equiv 1 \pmod{p}$.

In order to proceed to elliptic curve arithmetic we have to define data structures, which are able to represent points in both affine and projective coordinates.

```
**structure** $point[@X, @Y] <=$
  $inf$,
  $put(xc : @X, yc : @Y)$
```

```
**structure** $proj[@X, @Y, @Z] <=$
  $inf\_proj$,
  $put\_proj(xp : @X, yp : @Y, zp : @Z)$
```

Even though there is a dedicated value to represent infinity point in Jacobian coordinates, namely $(1 : 1 : 0)$, we decided to define explicit constants for representing infinity points in both affine and projective spaces for demonstration (*inf* and *inf _proj* respectively). Doubling returns the infinity point only if the same infinity point is its argument ($\nexists P : P \in E(K) \wedge P = -P$). So doubling of the infinity point and doubling of any other point can be viewed as two separate operations in both affine and projective coordinates. It is also possible to employ these dedicated infinity constants in addition operations if the following special cases are considered separately:

- If P and Q are added to each other and either of them is by chance a dedicated constant for $\emptyset$, another argument is returned;
- If $Q = -P$, then the respective dedicated constant for $\emptyset$ is returned;
- Otherwise, the respective general addition algorithm is used.

When the required eliptic-curve arithmetic operations are defined, it is possible to formulate lemmas, which correspond to both statements of the target Theorem 1.

```
lemma $Equation\ 7 <= \forall p1 : point[\mathbb{N}, \mathbb{N}], a, b, p : \mathbb{N}$
if $\{p > 3,$
  if $\{\mathbb{P}(p),$
    if $\{a = subm(p, 3, p),$
        if $\{?0(b\ mod\ p), true,$
          if $\{?0(yc(p1)\ mod\ p), true,$
            if $\{?0(discriminant(a, b, p)), true,$
              if $\{belongs(p1, a, b, p),$
                $ec\_dbl(p1, a, b, p)= jac2aff(ec\_dbl\_jac(aff2jac(p1), a, b, p), p),$
              $true\}\}\}\},$
    $true\},$
  $true\},$
$true\}$
```

```
lemma $Equation\ 8 <= \forall p1 : proj[\mathbb{N}, \mathbb{N}, \mathbb{N}],$
$a, b, p : \mathbb{N}$
if $\{p > 3,$
  if $\{\mathbb{P}(p),$
    if $\{a = subm(p, 3, p),$
        if $\{?0(b\ mod\ p), true,$
          if $\{?0(yc(jac2aff(p1, p))\ mod\ p), true,$
            if $\{?0(discriminant(a, b, p)), true,$
              if $\{belongs(jac2aff(p1, p), a, b, p),$
                $ec\_dbl(jac2aff(p1, p), a, b, p) = jac2aff(ec\_dbl\_jac(p1, a, b, p), p),$
              $true\}\}\}\},$
    $true\},$
  $true\},$
$true\}$
```

Verification of the target lemmas (summarized by Theorem 1) proved quite laborious and demanded most of the transformations to be done manually. This fact was partly caused by a substantial number of complex preconditions besides the terms, which correspond to equations (7) and (8) directly. √eriFun's heuristics was always trying to unfold and simplify those preconditions, despite the fact that they were needed only in very specific situations (i.e. fraction cancellation). High complexity of both parts of the target equation was another challenge for the system's heuristics.

The strategy of verifying each lemma was quite similar. First of all, the defined doubling functions for both coordinate systems treat the case of infinity-point separately. So verification of both target statements started from application of case analysis, which splits the entire proof tree into the obvious infinity-point case, and the complex general case. After some automated simplification and purging projective Z-coordinate, the general case is split to the proof of two separate equations, which characterized two different ways of calculating x- and y-coordinates of the resulting point (i.e. in affine and projective coordinates).

Finally, each verification step was carried out manually, with the intention to match left and right clauses of each equation. Verification of statement (7) comprised nearly 100 manual operations, while verification of statement (8) comprised more than 160 manual operations. These figures describe only the operations directly contained by the proofs; there are more of them if the ones contained by auxiliary lemmas are taken into account.

5. Evaluation

We have proved that point doubling operations for elliptic curves, described by equation (2), always return identical results either performed in affine or Jacobian projective coordinates. Preparatory work towards the proof of this fact can be viewed as a gradual introduction of definitions and verification of facts, which are related to the following domains: natural-number arithmetic, basic modular arithmetic, finite fields (GF(p)), and elliptic curves over GF(p).

Most of natural-number arithmetic and basic modular arithmetic lemmas have been proved either automatically or with some assistance in case √eriFun's heuristics chose wrong proof strategy. A set of non-trivial lemmas was only needed to propagate the properties of natural-number arithmetic operations to their analogs from modular arithmetic.

After introduction of prime numbers, relevant lemmas and their proofs became more complex. This was partly caused by the fact that the proof heuristics was always trying to unfold the complex recursive definition of a prime number, instead of treating it as an integral entity, which is characterized by some already proved properties. It is also impossible to perform induction on prime numbers in proofs for obvious reasons. Verification of Fermat's little theorem demanded extensive base of sophisticated facts related to lists, including properties of permutations and the pigeonhole principle.

As soon as the target lemmas proved extremely complex for automated verification, √eriFun's next-rule heuristics was turned off completely. Verification was carried out in manual mode using only first-order transformations without application of inductions.

Verification of the both statements (7) and (8) required basically the same set of facts with several exceptions of very targeted simplification lemmas, which were proved separately in order to simplify the target terms. This fact supports the idea that the introduced set of lemmas and definitions can ease verification of theorems similar to Theorem 1 for other elliptic-curve algorithms and other types of projective coordinate systems.

6. Related Work

Results of the first major research devoted to verification of a public key encryption algorithm were published by Boyer and Moore in 1984 [1].

Some research on verification of symmetric block ciphers has also been carried out. Duan et al. [2] prove invertibility of several popular block-cipher algorithms using HOL-4 theorem prover.

Hurd et al. [4] use the same theorem prover to formalize elliptic curve theory in higher order logic. The authors intend to create a mechanized 'gold standard' for elliptic curve operations. They define data types to represent elliptic curve equations, sets of points, arithmetic operations on elliptic curve points, and elliptic curve groups. The theorem, which represents ElGamal encryption for elliptic curves, is also defined. However, as soon as no proofs are given yet, it is hard to assume to what extent these definitions would be suitable for automated theorem proving.

Théry [11] defines elliptic curve operations in the theorem prover Coq. He proves a set of theorems, which form the group law for elliptic curve points.

Unlike authors of [4] and [11], we focus exclusively on a narrow practical problem of proving equivalence of optimized elliptic curve operations and their original descriptions. We do not intend to create a concise set of definitions suitable for further verification of any arbitrary fact related to elliptic curves as done in [4]. The research presented in this paper is targeted to getting the first proof-of-concept results as fast as possible, which draws the first approximation of data types, algorithms, and facts, needed for solutions of other similar problems.

7. Conclusion

We have presented a feasible approach to do a formal proof for the correctness of algorithmic variants in the domain of elliptic curve cryptography. This approach has been tested on the formal verification of the projective coordinate system, an ECC variant which is optimized towards a minimal number of finite field inversions.

The proof has been implemented and performed with the interactive theorem prover √eriFun. During the proving process we figured out some weaknesses in its rule application heuristic where it tries to apply an induction step over prime numbers. Thus some manual interaction was required to apply the correct rules. This should be improved in a future work step to allow a further automation of the verification.

Most of the introduced lemmas, their proofs, and the mathematical definitions hold for all elliptic curves. Thus, they can be adopted and reused for other algorithmic ECC variants and their subsequent correctness proofs. This will reduce the effort to generate new, and highly optimized algorithm variants with a guaranteed functional correctness.

The source file for √`eriFun` 3.2.2 with all the proofs is available at `http://bit.tsure.ru/sites/default/files/docs/ecc.vf`; the distributive of √`eriFun` 3.2.2 can be requested at `http://www.verifun.org`.

References

1. R. S. Boyer and J. S. Moore. Proof checking the RSA public key encryption algorithm. *American Mathematical Monthly*, 91(3):181–189, 1984.

2. J. Duan, J. Hurd, G. Li, S. Owens, K. Slind, and J. Zhang. Functional correctness proofs of encryption algorithms. In G. Sutcliffe and A. Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 519–533. Springer, 2005.

3. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag, 2004.

4. J. Hurd, M. Gordon, and A. Fox. Formalized elliptic curve cryptography. High Confidence Software and Systems: HCSS 2006.

5. N. Koblitz. Elliptic curve cryptosytems. *Mathematics of Computation*, 48(177):203–209, 1987.

6. A. K. Lenstra and E. R. Verheul. The XTR public key system. *Lecture Notes in Computer Science*, 1880:1–19, 2000.

7. V. S. Miller. Use of elliptic curves in cryptography. In H. C. Williams, editor, *CRYPTO85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426, 1985.

8. A. Riazanov and A. Voronkov. The Design and Implementation of VAMPIRE. *AI Communications*, 15(2):91–110, 2002.

9. R. L. Rivest, A. Shamir, and L. Adleman. A method of obtaining digital signaturtes and public-key cryptosystems. *Communications of the ACM*, 21:120–126, Feb. 1978.

10. S. Schulz. System Abstract: E 0.81. In D. Basin and M. Rusinowitch, editors, *2nd International Joint Conference on Automated Reasoning (IJCAR)*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 223–228. Springer-Verlag, July 2004.

11. L. Théry. Proving the group law for elliptic curves formally. Technical Report 0330, INRIA, Sophia Antipolis, 2007.

12. C. Walther, M. Aderhold, and A. Schlosser. The $\mathcal{L}$ 1.0 Primer. Technical Report VFR 06/01, Programmiermethodik, Technische Universität Darmstadt, Germany, Apr. 2006.

13. C. Walther et al. √`eriFun`: A verifier for functional programs, 2006.

14. C. Walther and S. Schweitzer. A machine supported proof of the unique prime factorization theorem. In D. Haneberg, G. Schellhorn, and W. Reif, editors, *Proc. of the 5th Workshop on Tools for System Design and Verification (FM-TOOLS 2002)*, volume 2002-11, pages 39–45, Augsburg, 2002. Institut für Informatik, Universität Augsburg.

Information about the authors:

Maxim Anikeev, PhD, Associate Professor, IT-security department, Southern Federal University, Taganrog, Russia
Felix Madlener, Researcher, Integrated Circuits and Systems, Technische Universität Darmstadt, Germany
Andreas Schlosser, Researcher, Programming Methodology, Technische Universität Darmstadt, Germany
Sorin A. Huss, PhD, Professor, Integrated Circuits and Systems, Technische Universität Darmstadt, Germany
Christoph Walther, PhD, Professor, Programming Methodology, Technische Universität Darmstadt, Germany