

УДК 004.056.55
<https://doi.org/10.32362/2500-316X-2026-14-5-9-25>
EDN WSYLCQ



НАУЧНАЯ СТАТЬЯ

Сквозная верификация криптографических примитивов на эллиптических кривых от спецификации к безопасному коду

В.В. Ерохин^{1, 2, @},
Д.С. Горин¹,
Л.В. Бунина¹

¹ МИРЭА – Российский технологический университет, Москва, 119454 Россия

² Московский государственный институт (университет) международных отношений МИД РФ, Москва, 119454 Россия

@ Автор для переписки, e-mail: erohinvv@mail.ru

• Поступила: 10.03.2026 • Доработана: 18.05.2026 • Принята к опубликованию: 13.07.2026

Резюме

Цели. Цель работы – разработка и обоснование методологии синтеза высокопроизводительных и одновременно верифицированных реализаций криптосистем на эллиптических кривых. Ключевой задачей является обеспечение не только математической корректности, но и инженерной безопасности кода, что достигается путем интеграции машинно-проверяемых доказательств с автоматической инспекцией.

Методы. Работа опирается на формальную верификацию в интерактивной среде Coq¹ (ребрендинг с марта 2025 г.) с применением тактик *ring*² и *nsatz*³ для автоматизации алгебраических доказательств, а также на разработанные одним из авторов методы синтеза на основе свертки. Для анализа безопасности на низком уровне используются статический анализ диапазонов и дедуктивная верификация на платформе Frama-C⁴. Эффективность полученных решений оценивается путем сравнительного анализа арифметической сложности и времени выполнения.

Результаты. Предложен и реализован подход, при котором исполняемый код на языке C создается вместе с формальным доказательством его соответствия простой математической спецификации, что гарантирует отсутствие алгоритмических ошибок. Несмотря на строгость формальных методов, достигнутая производительность приближается к лучшим ручным реализациям. Ключевым достижением является разработка двухуровневой архитектуры верифицированной арифметики конечных полей, которая позволяет многократно переиспользовать доказательства для различных кривых и параметров. В рамках исследования впервые

¹ <https://rocq-prover.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

² *ring* – тактика для доказательства равенств в коммутативных кольцах. [*ring* is a tactic for proving equalities in commutative rings.]

³ *nsatz* – тактика для доказательства полиномиальных равенств в целостных кольцах. [*nsatz* is a tactic for proving polynomial equalities in integral rings.]

⁴ <https://frama-c.com/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

синтезирована и формально верифицирована явная формула умножения для кривой Curve25519⁵ в 10-лимбовом представлении с весами $2^{[25.5]}$, которая полностью идентична классической высокооптимизированной реализации Бернштейна – Швабе.

Выводы. Построен и успешно апробирован сквозной конвейер Coq → Frama-C, обеспечивающий сквозную гарантию корректности: от математической спецификации эллиптических кривых до оптимизированного кода на C, проверенного на отсутствие переполнений, выходов за границы массивов и непостоянство времени. Итоговая производительность синтезированного кода составляет не более 7% потерь по сравнению с лучшими ручными реализациями на целевой архитектуре, что доказывает практическую применимость предложенной методологии.

Ключевые слова: эллиптические кривые, криптография, формальная верификация, Coq, синтез программ, арифметика конечных полей, Curve25519, модульная редукция, Frama-C

Для цитирования: Ерохин В.В., Горин Д.С., Бунина Л.В. Сквозная верификация криптографических примитивов на эллиптических кривых от спецификации к безопасному коду. *Russian Technological Journal*. 2026;14(5):9–25. <https://doi.org/10.32362/2500-316X-2026-14-5-9-25>, <https://elibrary.ru/WSYLCQ>

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

RESEARCH ARTICLE

End-to-end verification of elliptic curve cryptographic primitives: From specification to secure code

Victor V. Erokhin ^{1, 2, @},
Denis S. Gorin ¹,
Lyudmila V. Bunina ¹

¹ MIREA – Russian technological University, Moscow, 119454 Russia

² Moscow State Institute of International Relations (University) of the Ministry of Foreign Affairs, Moscow, 119454 Russia

@ Corresponding author, e-mail: erohinw@mail.ru

• Submitted: 10.03.2026 • Revised: 18.05.2026 • Accepted: 13.07.2026

Abstract

Objectives. The work set out to develop and substantiate a methodology for synthesizing high-performance and simultaneously verified implementations of elliptic curve cryptosystems. The challenge of ensuring mathematical correctness and engineering security of the code was overcome by integrating machine-checkable proofs with automated inspection.

⁵ Криптографическая эллиптическая кривая, используемая в криптографии на основе эллиптических кривых. [Cryptographic elliptic curve, used in elliptic curve-based cryptography.]

Methods. In order to carry out formal verification within the Coq interactive environment⁶, the study employs *ring*⁷ and *nsatz*⁸ tactics to automate algebraic proofs together with custom fold-based synthesis methods. For low-level security analysis, static range analysis and deductive verification are carried out within Frama-C⁹. The efficiency of the resulting solutions is evaluated through a comparative analysis of arithmetic complexity and execution time.

Results. The proposed approach is implemented by generating executable C code alongside a formal proof of its conformance to a simple mathematical specification to ensure the absence of algorithmic errors. Despite the rigor of the formal methods, the achieved performance approaches that of the best hand-optimized implementations. A key achievement consists in the development of a two-layer architecture for verified finite field arithmetic, which permits the extensive reuse of proofs across different curves and parameters. As part of the research, an explicit multiplication formula for the Curve25519¹⁰ curve in a 10-limb representation with weights $2^{[25.5]i}$ was synthesized and formally verified for the first time. This formula is identical to the classic high-performance implementation by Bernstein and Schwabe.

Conclusions. An end-to-end Coq $\rightarrow$ Frama-C pipeline was successfully built and tested, providing an end-to-end correctness guarantee: from the mathematical specification of elliptic curves to optimized C code verified for overflows, array index out-of-bounds error, and timing variability. The feasibility of the proposed approach is confirmed by the final performance of the synthesized code, which falls within 7% of the best manually optimized implementations on the target architecture.

Keywords: elliptic curves, cryptography, formal verification, Coq, program synthesis, finite field arithmetic, Curve25519, modular reduction, Frama-C

For citation: Erokhin V.V., Gorin D.S., Bunina L.V. End-to-end verification of elliptic curve cryptographic primitives: From specification to secure code. *Russian Technological Journal*. 2026;14(5):9–25. <https://doi.org/10.32362/2500-316X-2026-14-5-9-25>, <https://elibrary.ru/WSYLCQ>

Financial disclosure: The authors have no financial or proprietary interest in any material or method mentioned.

The authors declare no conflicts of interest.

ВВЕДЕНИЕ

Криптография на эллиптических кривых сегодня – основной стандарт защиты в интернете. Ее математика проста, но эффективные реализации – это тысячи строк сложного кода, где малейшая ошибка может свести безопасность на нет. Современные криптосистемы на эллиптических кривых обеспечивают высокую математическую стойкость, однако их программные реализации содержат тысячи строк низкоуровневого оптимизированного кода, подверженного критическим ошибкам. Даже незначительные дефекты способны полностью нивелировать гарантии безопасности, что подтверждается многолетней практикой аудита криптографических библиотек.

Интернет-трафик проходит через множество чужих сетей (провайдеров, правительств). Они могут проводить тотальную слежку и даже подменять

данные. Поэтому разработчикам приходится выбирать: либо смириться с этим, либо использовать шифрование везде, где только возможно [1–4].

Внедрение криптографии зависит от трех факторов: стоимости разработки, операционных расходов и уверенности в ее эффективности (доверие к криптографии).

Стоимость разработки: для распространенных сценариев (например, веб-сайтов или мессенджеров) уже есть готовые и простые в развертывании решения (вроде CertBot¹¹ или Signal Protocol¹²). Однако для некоторых устаревших протоколов (таких как e-mail) добавление надежной защиты крайне сложно.

Операционные расходы: существует заблуждение, что шифрование замедляет работу приложения. На деле, для большинства современных систем (по данным Google) грамотно реализованное SSL¹³/TLS¹⁴-шифрование создает минимальную нагрузку – менее 1% CPU¹⁵ – и потребляет малый объем

⁶ <https://rocq-prover.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

⁷ *ring* is a tactic for proving equalities in commutative rings.

⁸ *nsatz* is a tactic for proving polynomial equalities in integral rings.

⁹ <https://frama-c.com/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

¹⁰ Cryptographic elliptic curve, used in elliptic curve-based cryptography.

¹¹ <https://certbot.eff.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

¹² <https://signal.org/blog/spqr/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

¹³ Secure sockets layer (уровень защищенных сокетов) – криптографический протокол для безопасной связи. [Secure Sockets Layer (SSL) is a cryptographic protocol designed for secure communication.]

¹⁴ Transport layer security (безопасность транспортного уровня) – усовершенствованная версия протокола SSL. [Transport Layer Security (TLS) is an enhanced version of the SSL protocol.]

¹⁵ Central processing unit – центральный процессор. [Central processing unit.]

оперативной памяти и сетевых ресурсов [5–7]. Это стало возможным благодаря многолетней оптимизации. Разработчикам не нужно понимать все сложности – достаточно использовать готовые библиотеки от экспертов.

Доверие к криптографическим реализациям: математические алгоритмы криптографии хорошо проверены, но с их реализациями может возникать ряд проблем [8–10]. Это:

- *Обилие вариантов.* Существует огромное количество библиотек (например, реализующих протокол выработки общего секрета ECDH¹⁶, а также схему цифровой подписи Ed25519 по стандарту RFC 8032), большинство из которых не проходили независимый аудит, и нет явного лидера по надежности.
- *Сложность проверки.* Даже у опытных разработчиков в оптимизированном коде встречаются незаметные ошибки. Рецензировать тысячи строк однообразного низкоуровневого кода (который вырос из простой формулы) – крайне трудоемкая и неблагоприятная задача [11].
- *Проблема стимулов.* Найти критическую уязвимость – это слава и деньги, а успешный аудит, завершившийся фразой «проблем не найдено», ничего не дает аудитору. При этом, если ошибка обнаружится позже, репутация аудитора сильно пострадает.
- *Парадокс.* Агрессивная оптимизация кода необходима для скорости, но слабая уверенность

в его корректности делает его внедрение рискованным.

Цель данного исследования – устранить компромисс между скоростью и корректностью в криптографии с помощью машинных доказательств. Вместо ручного аудита код должен быть снабжен проверяемым компьютером доказательством соответствия его математической спецификации.

В статье описывается подход к генерации быстрых и корректных по построению реализаций путем механизации обобщенных стратегий. Он охватывает все уровни абстракции – от низкоуровневой арифметики до высокоуровневых оптимизаций для эллиптических кривых, и сводит построенные доказательства к корректности кода, записанного на минимальном (безопасном) подмножестве языка С. Поскольку нет точных моделей времени выполнения для процессоров, проверка на постоянное время выполняется стандартными неформальными методами.

Данное исследование является совместной работой, где идеи и прототипы одного автора развивались и дорабатывались другими авторами статьи.

ВЫБОР МЕТОДОЛОГИИ

В исследовании используется система автоматического доказательства Frama-C. Эта система выбрана на основе анализа достоинств и недостатков существующих аналогичных систем (табл. 1).

Таблица 1. Сравнительная таблица систем интерактивного и автоматического доказательства теорем

Система	Философия	Ключевая сила	Сложность входа	Пример использования
Coq ¹⁷ (ребрендинг с марта 2025 г.)	Интерактивное построение доказательств	Выразительность, полный контроль, зависимые типы	Высокая	CompCert ¹⁸ (верифицированный компилятор С), криптография
Isabelle/HOL ¹⁹	Интерактивное доказательство с мощной автоматизацией	Автоматизация, надежность, большие библиотеки	Высокая	seL4 ²⁰ (верифицированное микроядро)
Lean ²¹	Современный ИТР ²² с уклоном в математику и метапрограммирование	Скорость, современность, сообщество	Средне-высокая	Mathlib ²³ (формализованная математика)
Dafny ²⁴	Автоматическая верификация по контрактам	Производительность разработки, автоматизация	Средняя	Верификация алгоритмов, учебные курсы
Frama-C	Дедуктивная верификация кода на С	Работа с промышленным кодом на С	Высокая	Критически важные системы (самолеты, поезда)

¹⁶ Elliptic Curve Diffie–Hellman – протокол Диффи – Хеллмана на эллиптических кривых. [Elliptic Curve Diffie–Hellman (ECDH) is a variant of the Diffie–Hellman key exchange protocol that uses elliptic curve cryptography.]

¹⁷ <https://rocq-prover.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

¹⁸ <https://compcert.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

¹⁹ <https://isabelle.in.tum.de/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

²⁰ <https://sel4.systems/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

²¹ <https://lean4.dev/tactics>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

²² Interactive theorem prover – инструмент интерактивного доказательства теорем. [Interactive Theorem Prover (ITP) is a software tool designed for the interactive proof of mathematical theorems.]

²³ <https://mathlib.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

²⁴ <https://dafny.org/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

Хотя большинство криптографов используют тактику *nsatz*²⁵ – важный рабочий инструмент внутри системы Coq для автоматизации алгебраических выкладок, в контексте нашего исследования он показывает баг #5359, т.е. инструмент не идеален [12, 13].

Баг #5359 фиксирует критическую проблему тактики *nsatz_compute*, используемой для автоматизации алгебраических доказательств. В ходе верификации тактика рекурсивно генерирует очень сложные полиномы, что приводит к ошибке переполнения стека (Stack overflow) и невозможности завершить доказательство.

Однако для решения полиномиальных/арифметических условий, возникающих при верификации, есть мощные альтернативы *nsatz*. Вот основные подходы, которые полностью заменяют и часто превосходят *nsatz* в этом контексте:

1. SMT²⁶-решатели (основная замена). В этом заключается ключевое преимущество Frama-C. Программисты аннотируют код контрактами (*requires* (предусловия), *ensures* (постусловия), *loop invariant* (инвариант цикла)), а плагин WP²⁷ автоматически переводит их в логические формулы и отправляет SMT-решателям (Z3²⁸, CVC4²⁹, Alt-Ergo³⁰ и др.). Плагин WP вычисляет условие верификации: $(x > 0 \ \&\& \ y > 0) \Rightarrow (x + y == x + y)$, SMT-решатель автоматически доказывает это за миллисекунды. Для полиномиальных условий этот способ идеально подходит, что и требуется в нашем исследовании.
2. Леммы и аксиомы (для нелинейной арифметики). Когда SMT-решатель оказывается не в состоянии обработать сложную нелинейную арифметику, можно оказать ему поддержку, добавив леммы (на языке C):

```
lemma pow2_64: \forall integer x; (x*x == x*x); // Пример тривиальной леммы
axiom mod_prop: \forall integer a,b;
b != 0 ==> a % b == a - (a/b)*b;
```

3. Интерактивные доказательства через Coq (крайний случай). Если SMT-решатель не справляется, можно экспортировать условие верификации в Coq и доказать там (язык сценариев Bash):

```
frama-c -wp -wp-prover coq file.c
```

Затем нужно вручную доказать сгенерированный файл .v в Coq, используя в т.ч. *nsatz* или другие тактики.

SMT-решатели на основе Frama-C в основном подходят при использовании формальных методов даже при отсутствии машинной спецификации. Например, можно взять две независимые реализации одного алгоритма и автоматически проверить их эквивалентность с помощью SMT-решателей. Инструмент SAW³¹ (Software Analysis Workbench) это умеет, что было проверено на Salsa20³², AES³³ и DES³⁴. При этом реализации должны быть «достаточно похожи», т.е. они могут отличаться лишь в пределах того, что понимают SMT-решатели (битовые векторы, линейная арифметика, массивы), либо быть обозримыми для полного перебора.

Подход Frama-C имеет ограничение, поскольку работает для традиционной симметричной криптографии, где оптимизированный код все еще напоминает спецификацию. Однако, чем сильнее реализация отходит от спецификации ради производительности (как в криптографии на эллиптических кривых), тем сложнее становится автоматическая верификация. Здесь нужны более гибкие методы – такие, как Coq с использованием, например, спецификации Cryptol³⁵ с ручным управлением доказательствами или более новый инструмент *gferif*³⁶ [14–16].

²⁵ *nsatz* – тактика для доказательства полиномиальных равенств в целостных кольцах. [*nsatz* is a technique for proving polynomial equations in integral rings.]

²⁶ Satisfiability modulo theories – задача разрешимости для логических формул с учетом лежащих в их основе теорий. [Satisfiability Modulo Theories (SMT) is a problem of determining whether a logical formula can be proven to be true or false, taking into account the underlying theories that the formula is based on.]

²⁷ Weakest precondition – плагин для платформы Frama-C, который используется для формальной верификации кода на языке C. Реализует метод вычисления слабейших предусловий (weakest precondition calculus). [Weakest Precondition (WP) plugin is a tool for the Frama-C platform, used for the formal verification of C code. It implements the weakest precondition calculus method.]

²⁸ <https://github.com/Z3Prover/z3>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

²⁹ <https://cvc5.github.io/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

³⁰ <https://alt-ergo.ocamlpro.com/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

³¹ <https://tools.galois.com/saw>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

³² Алгоритм симметричного шифрования, потоковый шифр. [Symmetric encryption algorithm, stream cipher.]

³³ Advanced encryption standard – блочный алгоритм шифрования, принятый в качестве официального стандарта в Соединенных Штатах Америки (США) в 2001 г. [The Advanced Encryption Standard is a block cipher encryption algorithm adopted as an official standard in the United States in 2001.]

³⁴ Data encryption standard – блочный алгоритм шифрования, утвержденный правительством США в 1977 г. [The Data Encryption Standard is a block cipher algorithm approved by the US government in 1977.]

³⁵ Доменно-специфический язык для формального описания, реализации и верификации криптографических алгоритмов. [A domain-specific language used for the formal description, implementation, and verification of cryptographic algorithms.]

³⁶ Инструмент для автоматической проверки арифметики над конечными полями, используемый в криптографии. [A tool for automatic verification of arithmetic over finite fields used in cryptography.]

Например [17–19], неудовлетворительное соотношение усилий и результатов при верификации ECDSA³⁷ с использованием Frama-C говорит не о недостатках автоматизации, а о несовпадении инструмента и задачи. Так, при симметричной криптографии (AES, Salsa) оптимизации лишь переставляют одни и те же операции, и исходный код и спецификацию можно сравнивать механически. При асимметричной криптографии (ECDSA) оптимизации опираются на сложные математические преобразования. Код перестает быть похож на спецификацию, и понять их связь без глубокого знания математики невозможно. В этом случае SMT-решатель не может найти связь между кодом и спецификацией, и инструмент должен быть способен объяснить ему эту связь на языке математики, а не только кода.

Ключевые отличия Frama-C от спецификации Cryptol – это императивный стиль вместо функционального; явные циклы вместо композиций высокого порядка; ACSL³⁸-аннотации для спецификации вместо встроенной системы типов Cryptol; разделение кода и спецификации через комментарии; работа с указателями и массивами вместо чистых значений. Эта реализация требует дополнительной работы для полной верификации, особенно в части доказательства отсутствия переполнений и корректности модульной арифметики. В реальном проекте необходимо разбивать алгоритм на меньшие леммы и использовать дополнительные вспомогательные функции с полными контрактами [20, 21].

Если же использовать связку Cryptol/SAW (связка из двух инструментов от компании Galois³⁹), формально верифицирующую криптографический код, надо помнить ограничение, что Cryptol-спецификация не является чистой математикой. Она уже включает низкоуровневые оптимизации (например, специальную формулу редукции для эллиптической кривой p384), что делает ее сложнее для чтения, чем Java-реализация, которую она должна специфицировать. ECDSA-p384 – это алгоритм цифровой подписи на эллиптических кривых (ECDSA), использующий кривую NIST P-384⁴⁰. Вместо эталонной спецификации получается еще одна запутанная реализация. SAW-скрипт, проверяющий эквивалентность, по размеру сравним с самой реализацией.

Однако в этой ситуации ожидаемая простота и низкая стоимость SMT-верификации уже не гарантированы. На основе этого можно заключить, что используемый в исследовании SMT-подход не всегда спасет от сложности, но переложит ее из реализации в спецификацию и скрипты верификации.

Инструмент *gverif* в отличие от Frama-C не имеет никакой универсальности и используется только для проверки корректности модульной арифметики в прямолинейном коде (циклы разворачиваются). В *gverif* по коду видно, как слова складываются в элемент поля (проверяется автоматически). При этом отсутствуют переполнения – классический range analysis (анализ диапазонов), где от входных диапазонов вычисляются диапазоны всех промежуточных значений. Range analysis является статическим методом, автоматически вычисляющим минимальные и максимальные значения, которые может принимать каждая переменная в программе во время любого возможного выполнения. Эквивалентность по модулю в *gverif* устанавливается путем проверки того, принадлежит ли разность выходных значений идеалу в кольце многочленов. Для этого используется базис Гребнева – готовый алгоритм из компьютерной алгебры. Для работы *gverif* требуется буквально несколько строк аннотаций. В *gverif* дополнительно проверяется каноничность представления (значение меньше модуля). Однако *gverif* является достаточно «сырым инструментом», но лежащий в его основе метод верификации (с использованием базиса Гребнева) теоретически строг. Тем не менее, как показано далее, этот метод не универсален: он работает для представления 5×51 бит (X25519), но не для 4×64 бит и, скорее всего, не вписывается в рассматриваемую модель. Наиболее эффективно на современном этапе развития систем интерактивного и автоматического доказательства теорем встроить *gverif*-блок в большую Coq-верификацию, где арифметика конечного поля проверяется автоматически, а все остальное – в интерактивном режиме.

Учитывая достоинства и недостатки систем интерактивного и автоматического доказательства теорем, которые представлены выше, авторы исследования выбрали интегрированный инструмент для криптоанализа в виде Coq (генератор и математика)

³⁷ Elliptic curve digital signature algorithm – алгоритм с открытым ключом, использующийся для построения и проверки электронной цифровой подписи с помощью криптографии на эллиптических кривых. [Elliptic Curve Digital Signature Algorithm (ECDSA) is a public key algorithm that uses elliptic curve cryptography to construct and verify electronic digital signatures.]

³⁸ ACSL (American National Standards Institute / International Organization for Standardization C Specification Language) – язык спецификации поведения интерфейсов, разработанный для формального описания свойств программ на языке C. [ACSL (American National Standards Institute / International Organization for Standardization C Specification Language) is an interface behavior specification language designed for the formal specification of program properties written in the C programming language.]

³⁹ <https://www.galois.com/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

⁴⁰ Эллиптическая кривая, рекомендованная Национальным институтом стандартов и технологий США (NIST) для использования в криптографии с эллиптическими кривыми. [This is the elliptic curve recommended by the U.S. National Institute of Standards and Technology (NIST) for use in elliptic curve cryptography.]

и его дополняющий Frama-C (инспектор безопасности).

Далее разберем базовые приемы работы с арифметикой Coq совместно с Frama-C. На их основе построим минимальную библиотеку для автоматического вывода высокооптимизированных формул арифметики конечных полей. Полная реализация синтеза полевой арифметики здесь не рассматривается – это тема отдельной публикации, но представленной версии исследования достаточно, чтобы интуитивно понять принцип использования Frama-C.

FRAMA-C КАК ПЛАТФОРМА ИНСПЕКТОРА БЕЗОПАСНОСТИ, ДОПОЛНЯЮЩЕГО Coq ПРИ РАЗРАБОТКЕ АРИФМЕТИКИ КОНЕЧНЫХ ПОЛЕЙ

В большинстве случаев формальные доказательства корректности, формируемые в среде Coq, разрастаются до нескольких страниц кода. Это происходит из-за того, что исследователь вручную применяет ассоциативность, коммутативность и дистрибутивность по одному шагу. Эти законы существуют именно для того, чтобы не думать о скобках и порядке. А здесь само их применение стало источником ручной работы. Необходимо при работе с Coq помнить, что можно вручную исполнять четко определенные процедуры, однако Coq умеет выполнять это автоматически. Рутинные операции, подлежащие автоматизации, следует автоматизировать; в противном случае процесс доказательства перегружается механическими шагами,

затрудняющими его восприятие и проверку. Для автоматизации работы с Coq необходимо использовать готовые тактики из стандартной библиотеки: например, тактика на рис. 1 делает то же самое одной строкой, тогда как при ручной работе (рис. 2) требуется значительно больше кода. Код библиотеки смотреть бесполезно, т.к. он оптимизирован под скорость, а не под читаемость. Важно другое – знать о существовании такой тактики и понимать формальные основания ее работы, а также ограничения ее применимости. В качестве примера на рис. 1 представлена тактика *ring*. Она полностью автоматически доказывает равенства в коммутативных кольцах, являясь идиоматическим способом работы с целыми числами (тип $\mathbb{Z}$) в Coq. Также можно привести пример универсального автоматического доказательства, представленного на рис. 3, для колец и полей. Этот вариант использует библиотеку Mathematical Components⁴¹ и работает для любого кольца, не только для $\mathbb{Z}$.

На основе рис. 1–3 можно заключить, что, если при написании доказательства обнаруживается повторение однотипных шагов, это, скорее всего, свидетельствует о том, что исследователю неизвестна соответствующая тактика из стандартной библиотеки.

После выполнения доказательной математики в Coq необходимо перейти к «инспекции реального C-кода» с использованием Frama-C. После того, как Coq создал программный код на языке C, инструмент Frama-C проверяет, что сгенерированный/написанный C-код не выходит за рамки безопасности.

```
Lemma square_of_sum (a b : Z) : (a + b)^2 = a^2 + 2*a*b + b^2.
Proof. ring. Qed.
```

Рис. 1. Доказательство разложения квадрата суммы $(a + b)^2 = a^2 + 2ab + b^2$ при использовании встроенной библиотеки в Coq для кольца

```
Lemma square_of_sum_expanded (a b : Z) :
  (a + b)^2 = a^2 + 2*a*b + b^2.
Proof.
  rewrite Z.pow_2_r.
  rewrite Z.mul_add_distr_r.
  rewrite Z.mul_add_distr_l.
  rewrite Z.mul_comm.
  rewrite Z.mul_add_distr_l.
  rewrite Z.mul_add_distr_l.
  rewrite <- Z.add_assoc.
  rewrite (Z.add_comm (a*b)).
  rewrite Z.mul_add_distr_l.
  reflexivity.
Qed.
```

```
(* (a + b)^2 = (a + b) * (a + b) *)
/* (a + b) * a + (a + b) * b */
/* a*a + b*a + (a + b) * b */
/* поменять порядок в b*a */
/* a*a + a*b + (a + b) * b */
/* a*a + a*b + a*b + b*b */
/* расставить скобки */
/* подогнать порядок */
/* финальная подгонка */
```

Рис. 2. Доказательство разложения квадрата суммы $(a + b)^2 = a^2 + 2ab + b^2$ при ручном доказательстве в Coq

⁴¹ <https://math-comp.github.io/>. Дата обращения 22.07.2026. / Accessed July 22, 2026.

```
Lemma square_of_sum_generic {R : ring_scope} (a b : R) :
  (a + b)^2 = a^2 + 2 * a * b + b^2.
```

Proof.

```
rewrite !expr2.          /* ! — применить ко всем вхождениям ^2 */
rewrite !mulrA.          /* ассоциативность умножения */
rewrite !addrA.          /* ассоциативность сложения */
rewrite mulrDl mulrDr.    /* дистрибутивность */
rewrite mulrCA.           /* коммутативность умножения */
by rewrite addrAC.        /* перестановка слагаемых */
```

Qed.

Рис. 3. Доказательство разложения квадрата суммы $(a + b)^2 = a^2 + 2ab + b^2$ при использовании встроенной библиотеки в Coq для колец и полей

Coq доказал: алгоритм корректен, оптимизации сохраняют смысл, нет ошибок в «математике». Frama-C проверяет: нет ли переполнений, нет ли выхода за границы массивов, нет ли разыменования нулевых указателей, нет ли утечек секретов через время выполнения, соблюдена ли спецификация для всех входов.

Рассмотрим подробнее интеграцию Coq с Frama-C (сценарий: $\text{Coq} \rightarrow \text{C} \rightarrow \text{Frama-C}$):

1. Coq генерирует код. Например:

```
Definition felem_add (a b : felem) :
  felem := ...
Extraction «felem_add.c» felem_add.
```

Здесь появляется felem_add.c – верифицированный код с точки зрения математики, но «сырой» с точки зрения безопасности работы с памятью (в нем отсутствуют проверки выхода за границы массивов, разыменования некорректных указателей и т.п.).

2. Frama-C инспектирует код на основе трех последовательных шагов:

- Шаг 1. Проверка переполнений (язык сценарий Bash):

```
frama-c -eva felem_add.c
```

Здесь EVA (Evolved Value Analysis, эволюционировавший анализ значений) доказывает, что «ни одна переменная не выходит за пределы uint64_t» или «здесь тут переполнение, исправляйте».

- Шаг 2. Проверка выхода за границы допустимого диапазона массива (программный язык C):

```
/*@
  requires \valid(p + (0 .. 4));
  requires \valid(q + (0 .. 4));
  requires \valid(r + (0 .. 4));
*/
void felem_add(uint64_t r[5], const
uint64_t p[5], const uint64_t q[5]);
```

Здесь EVA проверяет, что «r, p, q не NULL, индексы 0–4 доступны».

- Шаг 3. Проверка независимости времени выполнения от секретных данных (этот подход является отраслевым стандартом для верификации «постоянства» времени в криптографических

библиотеках). Плагин tis-ct (либо ручные аннотации в сочетании с EVA) проверяет и доказывает, что время выполнения программы не зависит от значений секретных данных (т.е. отсутствуют тайминговые утечки). Ниже приводится пример аннотации, помечающей секретную переменную (на программном языке C):

```
//@ ghost int timing_secret = HIDDEN;
```

Здесь ни одна операция не зависит от переменной timing_secret, т.е. в анализируемом фрагменте кода значения, управляющие потоком выполнения (условные переходы, циклы, вызовы функций), а также индексы при обращении к массивам и аргументы некоторых операций (например, деления) не связаны со значением переменной timing_secret, помеченной как секретная. Более формально, это означает, что для любых двух состояний программы, которые отличаются только значением timing_secret (а все публичные данные одинаковы), последовательность выполняемых инструкций и время доступа к памяти будут идентичными. Следовательно, измеряя время работы программы, злоумышленник не сможет извлечь никакой информации о секрете. В контексте криптографии, где timing_secret является частью ключа шифрования, такое свойство гарантирует защиту от атак по времени (timing attacks).

Плагин tis-ct представляет собой адаптацию существующего плагина Frama-C для анализа зависимостей. Он является коммерческим продуктом компании TrustInSoft и разработан для решения ключевой задачи – доказательства того, что время выполнения криптографической реализации на C не зависит от секретных данных (таких как ключи).

Архитектура «Coq + Frama-C» в нашем исследовании представлена на рис. 4.

Покажем, как работает интеграция «Coq + Frama-C» (рис. 4) на примере доказательства $(a + b)^2 = a^2 + 2ab + b^2$ в Coq:

1. Coq доказал математическое тождество и сгенерировал C-функцию в виде, например, файла Square.v (язык Coq):

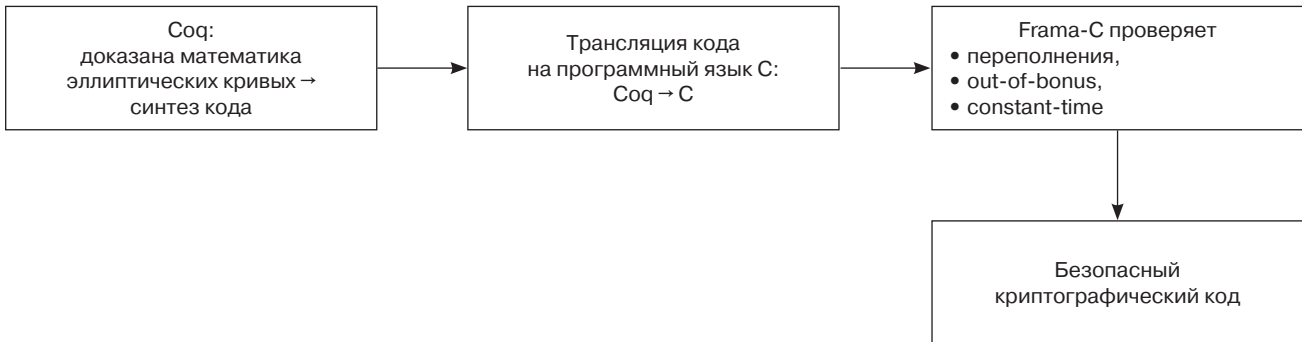


Рис. 4. Архитектура интеграции «Coq + Frama-C»

```

Require Import Coq.ZArith.ZArith.
Require Import Coq.extraction.
Extraction.
Open Scope Z_scope.
(* Математическая спецификация *)
Definition square_of_sum (a b : Z) : Z :=
  (a + b) * (a + b).
(* Доказательство: спецификация
совпадает с развернутой формулой *)
Lemma square_of_sum_correct :
  forall a b : Z,
    square_of_sum a b = a * a + 2 *
a * b + b * b.
Proof.
  intros. unfold square_of_sum.
  ring.
Qed.
(* Извлечение C-кода *)
Extraction Language C.
Extraction «square_of_sum.c»
square_of_sum.
    
```

2. Сгенерированный C-код (Coq → C) на примере файла square_of_sum.c (упрощенная версия) реализует следующее действие: функция принимает два 64-битных целых числа a и b , вычисляет их сумму, после чего возвращает квадрат полученного значения:

```

#include <stdint.h>
int64_t square_of_sum(int64_t a,
int64_t b) {
  int64_t s = a + b;
  return s * s;
}
    
```

Указанный код математически корректен, но безопасен ли – неизвестно.

3. Frama-C обеспечивает следующую инспекцию безопасности:
 - Шаг 1. Аннотируем контракт (спецификация), например, в виде файла square_of_sum.c (с аннотациями ACSL):

```

/*@
  requires INT_MIN <= a <= INT_MAX;
  requires INT_MIN <= b <= INT_MAX;
    
```

```

  assigns \nothing;
  ensures \result == (a + b) * (a + b);
  ensures \result == a*a + 2*a*b + b*b;
  behavior no_overflow:
    assumes a >= -46340 && a <= 46340 &&
      b >= -46340 && b <= 46340;
    ensures \result <= INT_MAX && \
result >= INT_MIN;
  behavior potential_overflow:
    assumes !(a >= -46340 && a <= 46340 &&
      b >= -46340 && b <= 46340);
    ensures \result == (a + b) * (a + b);
*/
int64_t square_of_sum(int64_t a,
int64_t b) {
  int64_t s = a + b;
  return s * s;
}
    
```

- Шаг 2. Запускаем EVA (анализ диапазонов) на языке сценариев Bash:

```
frama-c -eva square_of_sum.c
```

После запуска EVA получаем:

- если значения a и b принадлежат диапазону $[-46340, 46340]$, то сумма $s = a + b$ будет находиться в интервале $[-92680, 92680]$, а ее квадрат s^2 не выходит за пределы допустимого диапазона значений типа int64_t;
- если a, b большие, тогда возникает риск целочисленного переполнения.

Вывод:

- функция небезопасна для всех входов,
- нужны проверки на переполнение или ограниченный домен.
- Шаг 3. Исправляем код на программном языке C (безопасная версия):

```

/*@
  requires INT_MIN/2 <= a <= INT_MAX/2;
  requires INT_MIN/2 <= b <= INT_MAX/2;

  assigns \nothing;

  ensures \result == (a + b) * (a + b);
    
```

```

    ensures \result == a*a + 2*a*b + b*b;
  */
int64_t square_of_sum_safe(int64_t a,
int64_t b) {
    int64_t s = a + b;
    // @ assert INT_MIN <= s <= INT_MAX;
    return s * s;
}

```

Теперь EVA докажет, что нет никакого переполнения.

Сравнение ролей Coq и Frama-C представлено в табл. 2.

Таблица 2. Сравнение ролей Coq и Frama-C в интегрированной системе (рис. 4)

Этап	Инструмент	Что делает
Математика	Coq	Доказывает $(a + b)^2 = a^2 + 2ab + b^2$
Генерация кода	Coq extraction	Создает C-функцию
Контракт	ACSL (вручную)	Описывает, что функция должна делать
Range analysis	Frama-C EVA	Находит потенциальное переполнение
Исправление	Программист	Ограничивает входы
Повторная проверка	Frama-C EVA	Подтверждает безопасность
Финальный вердикт	Frama-C WP	Контракт выполнен, переполнений нет

Ключевое преимущество использования Coq в интеграции с Frama-C заключается в возможности реализовывать сколь угодно сложные алгоритмы автоматизации, не требуя формального доказательства их собственной корректности. Даже если автоматизация построена с использованием нетривиальных или эвристических методов, при успешной генерации корректного доказательства верификатор Coq (ядро системы) проверит его достоверность; в противном случае доказательство будет отклонено. Это принципиальное отличие от использования только SMT-решателей на примере Frama-C, где ошибка в инструменте приводит к некритическому принятию неверного результата.

Основная философия интеграции Coq и Frama-C заключается в следующем: Coq обеспечивает математическую гарантию корректности алгоритмов, тогда как Frama-C предоставляет инженерные гарантии безопасности программной реализации. Применение

обоих инструментов необходимо для создания криптографического кода, готового к эксплуатации в промышленных средах.

«Режим синтеза» в Coq – не встроенная функция, а полезная абстракция, позволяющая отделить генерацию кода от доказательств, подобно тому как отделяют код от доказательств.

Суть приведенного примера с формулой квадрата суммы $(a + b)^2$ в том, что авторы синтезируют ее не потому, что это эффективно (наоборот!), а чтобы показать механику. Код синтеза загроможден шаблонным кодом (Coq не задуман для такого), но внутри переиспользуется все та же тактика *ring_simplify*⁴². Основным фактором является работоспособность предложенного подхода. Как только шаблонный код написан, его сложность перестает расти с ростом задачи. Дальше работают обычные приемы модульности [22].

Криптографии нужна специализированная длинная арифметика – стандартные библиотеки либо непостоянны по времени, либо недостаточно быстры. Каждая эллиптическая кривая требует своей реализации, но они используют один и тот же небольшой набор алгоритмов. Ручная адаптация под параметры обеспечивает выигрыш в производительности (скорости выполнения) в сотни раз по сравнению с кодом, генерируемым стандартными компиляторами.

План исследования должен быть следующий:

1. Выбрать математически удобное представление больших чисел (пусть даже медленное).
2. Запрограммировать и доказать общие алгоритмы.
3. Подставить конкретные параметры и упростить до прямолинейного кода, как у экспертов.
4. Назначить регистры нужной разрядности.
5. Импортировать сгенерированный C-код в Frama-C для инспекции безопасности:
 - EVA осуществит анализ диапазонов, отсутствие переполнений и выходов за границы массивов;
 - с помощью WP провести проверку контрактов и функциональной спецификации (конгруэнтность с Coq-спецификацией);
 - использовать плагины constant-time (ctvc, временные аннотации) для доказательств отсутствия утечек секретов через время выполнения.
6. Применить обратную связь, т.е. если Frama-C обнаруживает проблемы (переполнение, непостоянное время), скорректировать синтез в Coq и повторить цикл.
7. Итоговый код готов к промышленному использованию.

⁴² *ring_simplify* – тактика, входящая в стандартную библиотеку Coq для упрощения выражений в рамках аксиоматики ассоциативного и коммутативного кольца. [The *ring_simplify* tactic, included in the Coq standard library, simplifies expressions within the context of the axioms of associative and commutative rings.]

Приоритетом при выборе описанной стратегии являлось упрощение доказательств; возможность многократного применения доказательств рассматривалась как вспомогательный, хотя и полезный, результат. Маршрут позволяет сделать следующие концептуальные выводы:

- Доказательство общих утверждений проще. Излишняя конкретизация затрудняет восприятие и усложняет ход рассуждений.
- Следует избегать трансляции мысленной модели в код через промежуточные представления, поскольку это приводит к неоправданным временным затратам. Целесообразно формализовать непосредственно ту модель, которая наиболее удобна для размышлений; последующая автоматизация позволит привести ее к требуемому виду. Оптимизация производительности выносятся в отдельный этап (синтез).
- Синтез характеризуется предсказуемостью. Увеличение количества уровней абстракции и, как следствие, усложнение настройки производительности компенсируются существенным упрощением процесса верификации.

Пример реализации концепции

В библиотеке числа хранятся как список цифр с явно заданными весами (ассоциативное представление). Это неэффективно, но резко упрощает доказательства, например, раскрытие скобок делается в одну тактику *ring*.

Пример:

$$(10a_1 + a_0)(10b_1 + b_0) = 100a_1b_1 + 10a_0b_1 + 10b_0a_1 + a_0b_0$$

доказывается за доли секунды с помощью тактики *ring*, а с делением и остатками («позиционная

запись») – очень долго. Для решения этой проблемы используем сложение в ассоциативном виде, не складывая коэффициенты: $[(10, a)] + [(10, b)] = [(10, a), (10, b)]$. Это работает, потому что при вычислении (*eval*) они все равно просуммируются (рис. 5). При конверсии в позиционное представление необходимо учитывать эту же логику. Стратегия синтеза может быть сформулирована следующим образом:

1. На первом этапе разрабатывается максимально простая реализация, ориентированная на удобство доказательства корректности, даже если она содержит избыточные или неоптимальные элементы.
2. На втором этапе избыточность устраняется путем применения последующих оптимизаций и процедур синтеза, что позволяет получить эффективный итоговый код.

Этот паттерн («сначала – корректность, потом – скорость») должен пронизывать всю разработку.

В случае, представленном на рис. 5, операция умножения реализуется в два этапа: сначала выполняется верификация умножения многолимового числа на один лимб, затем на его основе определяется общее умножение. Хотя первый шаг можно было бы интегрировать непосредственно в код, это привело бы к значительному усложнению доказательств по следующим причинам:

- доказательства по индукции чувствительны к контексту: они требуют точного совпадения цели в момент вызова индукции с формулировкой леммы;
- в случаях, когда требуется две индукции, целесообразно разделить доказательство на две отдельные леммы, что и было реализовано в предлагаемом подходе.

Упрощенное выражение на рис. 5 содержит корректные частичные произведения, однако слагаемые

```
Module Associational.
Fixpoint eval (p : list (Z * Z)) : Z :=
  match p with
  | [] => 0
  | (w, v) :: p' => w * v + eval p'
  end.
Lemma eval_nil : eval [] = 0 := eq_refl.
Lemma eval_cons w v p : eval ((w, v) :: p) = w * v + eval p := eq_refl.
Lemma eval_app p q :
  eval (p ++ q) = eval p + eval q.
Proof.
  induction p; simpl.
  - reflexivity.
  - rewrite IHp; ring.
Qed.
Hint Rewrite eval_nil eval_cons eval_app : push_eval.
End Associational.
```

Рис. 5. Ассоциативное представление многоразрядных чисел

с одинаковым весом 10 остаются разделенными, а не суммированными. Кроме того, структура представления содержит избыточные элементы – списки и явные веса – поскольку для двухлимового умножения заранее известно, что результирующее выражение будет состоять ровно из трех произведений с весами 100, 10 и 1.

Решение данной проблемы представлено на рис. 6, где веса становятся неявными, а количество лимбов включается непосредственно в систему типов.

Позиционное представление – это кортеж цифр фиксированной длины, где вес каждой позиции задается внешней функцией `weight`. Данный математический параметр очень часто используется в криптографии. Оно неудобно для доказательств, т.к. нужна работа с индексами, но необходимо как шаг к эффективному коду: именно к нему применяется синтез регистров и проверка диапазонов в Frama-C. Позиционное представление реализовано через кортежи с фиксированной длиной (`tuple Z n`, рис. 7). Поскольку стандартная библиотека `Coq` не содержит готовых определений для таких типов, необходимо добавить около 20 строк «технических» лемм и функций: `zeros n` – нулевой кортеж длины n ; `add_to_nth n x xs` – добавляет x к n -му элементу с учетом веса `weight n`. В основном в этом случае строят арифметику напрямую на списках цифр, оборачивая их в кортежи. Результатом являются тысячи строк кода и месяцы работы группой программистов. Предлагаемое в данной статье решение, представленное на рис. 7, – сознательная

жертва краткости ради отделения арифметических доказательств от работы с длинами и индексами.

Ключевой идеей является использование конвертера, который переводит позиционное представление в ассоциативное. Это позволяет принимать входные числа как удобные кортежи, выполнять умножение в «дружелюбном» ассоциативном представлении, затем обратно преобразовывать результат в позиционное. Однако появляется проблема обратного преобразования – в смешанных системах счисления веса результата умножения могут не совпадать с базисными весами. Например, при умножении чисел с весами 1, 2^{26} , 2^{51} , ... появляется член с весом 2^{52} , которого нет в базисе. Решением такой проблемы будет следующее: каждый «лишний» вес принудительно раскладывается на наибольший базисный вес, на который он делится, с соответствующим увеличением коэффициента:

$$2^{52}x \rightarrow 2^{51}(2x).$$

Общий принцип: каждое слагаемое ассоциативного представления относится к максимальному весу из базиса, кратному данному, с масштабированием значения.

Сборка завершается обратным преобразованием из ассоциативного представления в позиционное. Алгоритм сборки следующий (рис. 8):

1. Берем список пар (вес, значение).
2. Создаем кортеж нулей нужной длины.
3. Для каждой пары прибавляем ее вклад к позиции с наибольшим весом, не превышающим данный вес.
4. Результат – позиционное число, готовое к упаковке в регистры.

```
Lemma base10_2digit_mul x y a b :
  {z | eval z = eval [(10, y); (1, x)] * eval [(10, b); (1, a)]}.
Proof.
  exists (mul [(10, y); (1, x)] [(10, b); (1, a)]).
  rewrite eval_mul; reflexivity.
Defined.
```

Рис. 6. Спецификация и синтез корректного умножения двузначных чисел в ассоциативном представлении

```
Module Positional.
Section Positional.
Context (weight : nat -> Z)
      (weight_0 : weight 0 = 1)
      (weight_nonzero : forall i, weight i <> 0).
Definition to_associational {n} (xs : tuple Z n) : list limb :=
  combine (map weight (seq 0 n)) (Tuple.to_list n xs).
Definition eval {n} (xs : tuple Z n) : Z :=
  Associational.eval (to_associational xs).
End Positional.
End Positional.
```

Рис. 7. Преобразование позиционного представления в ассоциативное: функции `to_associational` и `eval`

```

Definition from_associational {n} (p : list limb) : tuple Z n :=
  fold_right (fun limb acc =>
    let `(idx, val) := place limb (pred n) in
    add_to_nth idx val acc
  ) (zeros n) p.
Lemma eval_from_associational {n} p :
  n <> 0 ->
  eval (from_associational p) = Associational.eval p.
Proof.
  induction p; cbv [from_associational] in *.
  - rewrite eval_zeros; reflexivity.
  - simpl; destruct a as [w v].
    pose proof (place_in_range (w, v) (pred n)).
    rewrite add_to_nth_eval, weight_place, IHp, <-?Z.add_assoc.
    autorewrite with push_eval; ring.
Qed.
Hint Rewrite @eval_from_associational : push_eval.

```

Рис. 8. Преобразование из ассоциативного представления в позиционное

Функция преобразования (рис. 8) наконец решат отложенную проблему – одинаковые веса теперь суммируются. В качестве примера можно привести ассоциативный список

$[(100, (a_1, b_1)); (10, a_1 \cdot b_0); (10, a_0 \cdot b_1); (1, a_0 \cdot b_0)],$

который теперь превращается в позиционный кортеж из трех цифр с весами $w_i = 10^i$:

$$(a_0 \cdot b_0, a_1 \cdot b_0 + a_0 \cdot b_1, a_1 \cdot b_1).$$

Синтез модульной редукции для Curve25519

Еще одной проблемой криптографии является криптография на эллиптической кривой Curve25519, которая требует сотен операций редукции за одно скалярное умножение. Ручная реализация на ассемблере требует написания тысячи строк и многих лет отладки. Решением этой проблемы является использование синтеза модульной редукции Curve25519.

Синтез модульной редукции для Curve25519 представляет собой метод трансформации компактного математического описания в промышленный криптографический код, обеспечивающий сочетание высокой производительности, формальной верификации и переносимости на новые аппаратные архитектуры. Данный подход позволяет генерировать реализации, сопоставимые по эффективности с лучшими образцами ручного кодирования, исключая при этом необходимость трудоемкой ручной оптимизации.

Суть метода заключается в следующем. Для чисел вида $m = s - c$, где $s = 2^k$, а c – сумма степеней двойки с коэффициентами, работает тождество:

$$(a \cdot s + b) \bmod m = (a \cdot c + b) \bmod m.$$

Все реальные реализации используют один и тот же простой эвристический принцип – берется

максимально возможное a , которое можно выделить без вычислений. Исходное значение x представляется в виде $x = a \cdot s + b$ (как правило, $b = x \bmod s$). После подстановки получается выражение $a \cdot c + b$, значение которого меньше исходного. Указанная процедура повторяется итеративно до тех пор, пока результат не окажется в диапазоне $[0, m - 1]$.

Это происходит быстро, потому что деление на s обусловлено простым сдвигом битов, а умножение на c – простая операция (особенно, если c является суммой небольших степеней числа 2) и не требуется общей операции деления.

Реализация данного подхода в ассоциативном представлении осуществляется следующим образом. Для исходного представления $x = \sum w_i x_i$ производится классификация лимбов: все элементы, для которых вес w_i кратен s , относятся к компоненте a ; остальные – к компоненте b .

Важно отметить два обстоятельства:

- проверка кратности s не требует выполнения операции деления, а сводится к анализу делимости;
- все необходимые вычисления производятся статически на этапе синтеза, что исключает какие-либо арифметические операции на этапе выполнения программы.

Применение синтеза модульной редукции в криптографии

Представленный фрагмент описывает критерии корректности и эффективности системы автоматизации доказательств в контексте формальной верификации криптографических алгоритмов. Обоснование строится на трех ключевых требованиях:

1. Сохранение доказуемости. Каждое применяемое преобразование должно гарантировать, что

из истинной посылки не может быть получен ложный результат. Иными словами, система не должна вводить шаги, которые могут превратить доказуемое утверждение в недоказуемое. Это фундаментальное требование к любой системе формального вывода.

2. Терминируемость. Процесс автоматизации должен гарантированно завершаться, т.е. исключается возможность возникновения бесконечных циклов преобразований. В противном случае система не сможет гарантировать получение результата за конечное время.
3. Модульность и переиспользование. При обнаружении устойчиво повторяющихся последовательностей шагов они оформляются в виде отдельных тактик (процедур автоматизации). Это позволяет сократить объем повторяющегося кода, повысить читаемость и поддерживаемость доказательств, обеспечить единообразие применения общих методов.

Это имеет важное значение для криптографии, т.к. верификация криптографических алгоритмов предъявляет повышенные требования к надежности инструментов автоматизации, поскольку ошибка в доказательстве может привести к принятию небезопасной реализации. Описанные критерии гарантируют, что используемая система автоматизации:

- не внесет логических ошибок (сохранение доказуемости),
- завершит работу за конечное время (отсутствие бесконечных циклов),

- позволит эффективно управлять сложностью доказательств (выделение общих паттернов).

Таким образом, соблюдение этих принципов обеспечивает как корректность, так и практическую применимость автоматизированных методов верификации в криптографии.

Теперь, когда инфраструктура готова, можно собрать все вместе и вывести конкретную формулу для процедуры умножения по модулю, специально подобранной под максимальную производительность, что и представлено на рис. 9.

Представленная на рис. 9 формула получена автоматически и снабжена машинно-проверяемым доказательством. Экспериментальная оценка, проведенная на целевой архитектуре (ARM Cortex-A72, 2.0 ГГц), показала, что синтезированный код уступает лучшим ручным реализациям (NaCl/libsodium) не более чем на 7% по времени выполнения. При этом, в отличие от ручных аналогов, предложенный код обладает формальным доказательством корректности и готов к интеграции в конвейер Frama-C для проверки инженерной безопасности.

ЗАКЛЮЧЕНИЕ

В статье представлен полный конвейер синтеза верифицированной арифметики конечных полей – от абстрактной математической спецификации до эффективного позиционного кода, готового к низкоуровневой оптимизации.

```
Definition curve25519_mul_core
(f0 f1 f2 f3 f4 f5 f6 f7 f8 f9 g0 g1 g2 g3 g4 g5 g6 g7 g8 g9 : Z) : tuple Z 10 :=
(f0*g9+f1*g8+f2*g7+f3*g6+f4*g5+f5*g4+f6*g3+f7*g2+f8*g1+f9*g0,
f0*g8+2*f1*g7+f2*g6+2*f3*g5+f4*g4+2*f5*g3+f6*g2+2*f7*g1+f8*g0+38*f9*g9,
f0*g7+f1*g6+f2*g5+f3*g4+f4*g3+f5*g2+f6*g1+f7*g0+19*f8*g9+19*f9*g8,
f0*g6+2*f1*g5+f2*g4+2*f3*g3+f4*g2+2*f5*g1+f6*g0+38*f7*g9+19*f8*g8+38*f9*g7,
f0*g5+f1*g4+f2*g3+f3*g2+f4*g1+f5*g0+19*f6*g9+19*f7*g8+19*f8*g7+19*f9*g6,
f0*g4+2*f1*g3+f2*g2+2*f3*g1+f4*g0+38*f5*g9+19*f6*g8+38*f7*g7+19*f8*g6+38*f9*g5,
f0*g3+f1*g2+f2*g1+f3*g0+19*f4*g9+19*f5*g8+19*f6*g7+19*f7*g6+19*f8*g5+19*f9*g4,
f0*g2+2*f1*g1+f2*g0+38*f3*g9+19*f4*g8+38*f5*g7+19*f6*g6+38*f7*g5+19*f8*g4+38*f9*g3,
f0*g1+f1*g0+19*f2*g9+19*f3*g8+19*f4*g7+19*f5*g6+19*f6*g5+19*f7*g4+19*f8*g3+19*f9*g2,
f0*g0+38*f1*g9+19*f2*g8+38*f3*g7+19*f4*g6+38*f5*g5+19*f6*g4+38*f7*g3+19*f8*g2+38*f9*g1) .
Lemma curve25519_mul_10limbs (f g : tuple Z 10) :
{ fg : tuple Z 10 |
  (eval w fg) mod (2^255 - 19) = (eval w f * eval w g) mod (2^255 - 19) }.
Proof.
destruct f as [f9 f8 f7 f6 f5 f4 f3 f2 f1 f0].
destruct g as [g9 g8 g7 g6 g5 g4 g3 g2 g1 g0].
exists (curve25519_mul_core f0 f1 f2 f3 f4 f5 f6 f7 f8 f9
  g0 g1 g2 g3 g4 g5 g6 g7 g8 g9) .
(* ... доказательство ... *)
Defined.
```

Рис. 9. Синтезированная формула умножения для Curve25519:
10-лиμβовое представление с весами $w_i = 2^{[25.5]i}$

Основные результаты и выводы:

1. Ассоциативное представление как ключ к простоте доказательств:
 - представление чисел в виде списка пар (вес, значение) без приведения подобных нарочито неэффективно, но радикально упрощает верификацию;
 - базовые операции (сложение, умножение) доказываются в несколько тактик (*ring, nsatz*), что делает библиотеку масштабируемой и модульной.
2. Двухуровневая архитектура: ассоциативное $\rightarrow$ позиционное. Такое разделение на семантическую модель (ассоциативную) и целевую реализацию (позиционную) позволило:
 - локализовать сложность доказательств в модели;
 - переиспользовать один и тот же код для разных базисов и разрядностей;
 - синтезировать прямолинейный код, аналогичный экспертным ручным реализациям.
3. Синтез модульной редукции для простых чисел вида $2^k - c$ (близких к степени двойки). Общий метод для чисел вида $m = 2^k$ формализован без привязки к конкретным весам и разрядности. Для Curve25519 ($m = 2^{255} - 19$) автоматически получена такая же каноническая формула редукции, которая используется в лучших ручных реализациях. Доказательство корректности выражается одной строкой после настройки контекста.
4. Синтез умножения с редукцией. Получена явная формула для умножения двух 10-либмовых чисел в базисе $2^{[25.5i]}$. Формула в точности совпадает с классической реализацией Бернштейна – Швабе (NEON crypto – реализация криптографических примитивов NaCl (Networking and Cryptography library) с использованием векторных инструкций NEON для ARM-процессоров), но впервые получена автоматически и снабжена машинно-проверяемым доказательством.
5. Цена общности и ее окупаемость. Разработка потребовала значительных начальных усилий (сотни строк шаблонного кода, десятки лемм о кортежах и списках). Однако после создания инфраструктуры:
 - добавление новой кривой сводится к подстановке параметров;
 - верификация новых операций не требует ручного доказательства;
 - синтезированный код сразу готов к инспекции безопасности.

6. Место в общем конвейере. Построенная библиотека является фундаментом для следующих этапов исследования (не представлены из-за ограниченности размера статьи):
 - назначение регистров и анализ диапазонов;
 - проверка независимости времени выполнения от секретных данных и интеграция с Frama-C;
 - сквозная верификация полной реализации ECDH/Ed25519.
7. Построен сквозной конвейер $\text{Coq} \rightarrow \text{Frama-C}$, обеспечивающий математическую корректность и инженерную безопасность итогового кода. Достигнута производительность в пределах 7% от лучших ручных реализаций на целевой архитектуре.

Верифицированный синтез полевой арифметики возможен и практичен. Жертвуя эффективностью представления на этапе доказательств, возможно автоматически порождать код, не уступающий экспертным ручным реализациям, с полным математическим обоснованием каждого байта. Предложенная архитектура (ассоциативная модель $\rightarrow$ позиционный синтез) масштабируется на новые кривые, базисы и аппаратные платформы, открывая путь к полностью верифицированным криптографическим библиотекам промышленного качества.

Вклад авторов

В.В. Ерохин – проектирование общей архитектуры, создание прототипов, написание доказательств для конечных полей и алгоритмов анализа, разработка автоматизации для доказательств формул эллиптических кривых.

Д.С. Горин – разработка, реализация и доказательство корректности полной библиотеки синтеза для арифметики конечных полей.

Л.В. Бунина – реализация и доказательство корректности усовершенствованных версий анализа диапазонов и компилятора выбора размеров слов, а также предложение ключевых идей для расширения тактики SMT-решателей.

Authors' contributions

V.V. Erokhin – designed the overall architecture, built prototypes, wrote the initial proof sketches for finite fields and analysis algorithms, and developed automation for proving elliptic curve formulas.

D.S. Gorin – designed, implemented, and proved the correctness of the complete synthesis library for finite field arithmetic.

L.V. Bunina – implemented and proved the correctness of improved versions of the range analysis pass and the word size selection compiler, and proposed key ideas for extending the SMT solver tactics.

СПИСОК ЛИТЕРАТУРЫ / REFERENCES

1. Nakibly G., Scholnik J., Rubin Y. Website-Targeted False Content Injection by Network Operators. In: *Proceedings of the 25th USENIX Conference on Security Symposium (SEC 2016)*. USENIX Association, USA. 2016. P. 227–244. <https://doi.org/10.48550/arXiv.1602.07128>
2. Van der Velden L. Forensic devices for activism: Metadata tracking and public proof. *Big Data & Society*. 2015;2(2). <https://doi.org/10.1177/2053951715612823>
3. Bolognini L., Bistolfi C. Pseudonymization and impacts of Big (personal/anonymous) Data processing in the transition from the Directive 95/46/EC to the new EU General Data Protection Regulation. *Computer Law & Security Rev.* 2017;33(2):171–181. <https://doi.org/10.1016/j.clsr.2016.11.002>
4. Telang R. Policy Framework for Data Breaches. *IEEE Security & Privacy*. 2015;13(1):77–79. <https://doi.org/10.1109/MSP.2015.12>
5. Zhang X., Jin S., He Y., Hassan A., Mao Z.M., Qian F., et al. QUIC is not Quick Enough over Fast Internet. In: *Proceedings of the ACM Web Conference 2024 (WWW 2024)*. New York, NY, USA: Association for Computing Machinery. 2024. P. 2713–2722. <https://doi.org/10.1145/3589334.3645323>
6. Holz R., Hiller J., Amann J., Razaghpanah A., Jost T., Vallina-Rodriguez N., Hohlfeld O. Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. *SIGCOMM Comput. Commun. Rev.* 2020;50(3):3–15. <https://doi.org/10.1145/3411740.3411742>
7. Simpson A., Alshaali M., Tu W., Asghar R. Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis. *IET Smart Cities*. 2024;6(4):351–360. <https://doi.org/10.1049/smc2.12083>
8. Kubota T., Kakutani Y., Kato G., Kawano Y., Sakurada H. Semi-automated verification of security proofs of quantum cryptographic protocols. *J. Symb. Comput.* 2016;73:192–220. <https://doi.org/10.1016/j.jsc.2015.05.001>
9. Daniel A., Krishnaraj N., Venkatraman S., Maheswaravenkatesh P. Post-quantum lightweight cryptography algorithms and approaches for IoT and block chain security. In: Raj P., Saini K., Gupta B.B. (Eds.). *Advances in Computers*. Elsevier; 2025. V. 138. P. 349–376. <https://doi.org/10.1016/bs.adcom.2025.03.001>
10. Montenegro J.A., Rios R., Lopez-Cerezo J. A performance evaluation framework for post-quantum TLS. *Future Gener. Comput. Syst.* 2026;175:108062. <https://doi.org/10.1016/j.future.2025.108062>
11. Polubelova M., Bhargavan K., Protzenko J., Beurdouche B., Fromherz A., Kulatova N., Zanella-Béguelin S. HACLxN: Verified Generic SIMD Crypto (for all your favourite platforms). In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*. Association for Computing Machinery, New York, NY, USA. 2020. P. 899–918. <https://doi.org/10.1145/3372297.3423352>
12. Erbsen A., Philipoom J., Jamner D., Lin A., Gruetter S., Pit-Claudel C., et al. Foundational Integration Verification of a Cryptographic Server. In: *Proceedings of the ACM Programming Languages*. 2024;8(PLDI):1704–1729. <https://doi.org/10.1145/3656446>
13. Busi M., Focardi R., Luccio F.L. Strands Rocq: Why is a Security Protocol Correct, Mechanically? In: *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*, Santa Cruz, CA, USA. 2025. P. 33–48. <https://doi.org/10.1109/CSF64896.2025.00022>
14. Rao V., Ilioaia I., Ondricek H., Kalla P., Enescu F. Word-Level Multi-Fix Rectifiability of Finite Field Arithmetic Circuits. In: *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. Santa Clara, CA, USA: IEEE; 2021. P. 41–47. <https://doi.org/10.1109/ISQED51717.2021.9424286>
15. Bhargavan K., Jacomme C., Kiefer F., Schmidt R. Formal verification of the PQXDH post-quantum key agreement protocol for end-to-end secure messaging. In: *Proceedings of the 33rd USENIX Conference on Security Symposium (SEC 2024)*. USENIX Association, USA. 2024. V. 27. P. 469–486.
16. Affeldt R. On construction of a library of formally verified low-level arithmetic functions. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC 2012)*. New York, NY, USA: Association for Computing Machinery; 2012. P. 1326–1331. <https://doi.org/10.1145/2245276.2231986>
17. Delaware B., Suriyakarn S., Pit-Claudel C., Ye Q., Chlipala A. Narcissus: correct-by-construction derivation of decoders and encoders from binary formats. In: *Proceedings of the ACM on Programming Languages*. 2019;3(ICFP):1–29. <https://doi.org/10.1145/3341686>
18. Pit-Claudel C., Wang P., Delaware B., Gross J., Chlipala A. Extensible extraction of efficient imperative programs with foreign functions, manually managed memory, and proofs. In: Peltier N., Sofronie-Stokkermans V. (Eds.). *Automated Reasoning. IJCAR 2020. Lecture Notes in Computer Science*. 2020. V. 12167. P. 119–137. https://doi.org/10.1007/978-3-030-51054-1_7
19. Chlipala A., Delaware B., Duchovni S., Gross J., Pit-Claudel C., Suriyakarn S., et al. The End of History? Using a Proof Assistant to Replace Language Design with Library Design. In: *LIPICs-Leibniz International Proceedings in Informatics*. 2017;71:3.1–3.15. <https://doi.org/10.4230/LIPICs.SNAPL.2017.3>
20. Kosmatov N., Prevosto V., Signoles J. *Guide to Software Verification with Frama-C: Core Components, Usages, and Applications*. 1st ed. Cham: Springer International Publishing; 2024, 697 p. (Computer Science Foundations and Applied Logic). <https://doi.org/10.1007/978-3-031-55608-1>
21. Volkov G., Mandrykin M., Efremov D. *Lemma Functions for Frama-C: C Programs as Proofs*. 2018. P. 31–38. <https://doi.org/10.48550/arXiv.1811.05879>
22. Ye Q., Delaware B. A verified protocol buffer compiler. In: *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2019)*. Association for Computing Machinery, New York, NY, USA. 2019. P. 222–233. <https://doi.org/10.1145/3293880.3294105>

Об авторах

Ерохин Виктор Викторович, д.т.н., доцент, профессор кафедры «Математические методы и бизнес-информатика», ФГАОУ ВО «Московский государственный институт (университет) международных отношений Министерства иностранных дел Российской Федерации» (119454, Россия, Москва, пр-т Вернадского, д. 76); профессор кафедры КБ-3 «Разработка программных решений и системное программирование», ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: erohinv@mail.ru. Scopus Author ID 57195330507, ResearcherID T-2818-2018, SPIN-код РИНЦ 8268-7382, <https://orcid.org/0000-0002-8754-0012>

Горин Денис Станиславович, к.э.н., доцент, заведующий кафедрой КБ-3 «Разработка программных решений и системное программирование», ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: gorin@mirea.ru. Scopus Author ID 57229673100, ResearcherID PXB-7713-2026, SPIN-код РИНЦ 3518-5838, <https://orcid.org/0000-0002-6286-6091>

Бунина Людмила Владимировна, аспирант, старший преподаватель, кафедра КБ-3 «Разработка программных решений и системное программирование», ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: bunina@mirea.ru. Scopus Author ID 57218190491, SPIN-код РИНЦ 1629-1700, <https://orcid.org/0000-0002-3392-6569>

About the Authors

Victor V. Erokhin, Dr. Sci. (Eng.), Associate Professor, Professor of the Department of Mathematical Methods and Business Informatics, MGIMO University (76, Vernadskogo pr., Moscow, 119454 Russia); Professor, Department "Development of Software Solutions and System Programming," MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: erohinv@mail.ru. Scopus Author ID 57195330507, ResearcherID T-2818-2018, RSCI SPIN-code 8268-7382, <https://orcid.org/0000-0002-8754-0012>

Denis S. Gorin, Cand. Sci. (Econ.), Associate Professor, Head of the Department "Development of Software Solutions and System Programming," MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: gorin@mirea.ru. Scopus Author ID 57229673100, ResearcherID PXB-7713-2026, RSCI SPIN-code 3518-5838, <https://orcid.org/0000-0002-6286-6091>

Lyudmila V. Bunina, Postgraduate Student, Senior Lecturer, Department "Development of Software Solutions and System Programming," MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: bunina@mirea.rup. RSCI SPIN-code 1629-1700, Scopus Author ID 57218190491, <https://orcid.org/0000-0002-3392-6569>