УДК 004.056.55:004.432.2 DOI 10.18372/2073-4751.87.21684

Котух Є.В. [1], д.т.н.
orcid.org/0000-0003-4997-620X,
e-mail: yevgenkotukh@gmail.com;

Шкляр О.І. [2],
orcid.org/0009-0001-6524-6357,
e-mail: oleh.shklyar@gmail.com,

ЗАСТОСУВАННЯ ТИПОБЕЗПЕЧНИХ АВТОМАТІВ У МОВІ OCAML ДЛЯ АВТЕНТИФІКОВАНОГО УЗГОДЖЕННЯ КЛЮЧІВ

[1]Воєнна академія імені Євгенія Березняка
[2]Національний університет «Київський авіаційний інститут»

Вступ

Значна частка критичних уразливостей у реалізаціях протоколів захищеного зв'язку є не помилками криптографічних примітивів, а помилками керування станом протоколу. Показовими є дефект "goto fail" у бібліотеці Apple SecureTransport (CVE-2014-1266), внаслідок якого перевірка підпису параметрів обміну ключами не виконувалась, та сімейство атак на автомат станів TLS (SMACK, FREAK, Early Finished), систематично виявлених у роботі [1] для OpenSSL, GnuTLS, NSS, JSSE та інших бібліотек. У всіх цих випадках реалізація приймала синтаксично коректне повідомлення у стані, в якому специфікація його не допускає, або переходила у стан "автентифіковано" без виконання відповідної перевірки. Спільною причиною є те, що множина допустимих переходів автомата існує лише у тексті специфікації та у тестах, а не в артефакті, який перевіряє машина.

Актуальність задачі посилюється переходом до TLS 1.3 [2] та поширенням протоколів сімейства Noise [3] у мережах з обмеженими ресурсами, VPN (WireGuard) та месенджерах, а також початком постквантової міграції, коли до автоматів рукостискання додаються гібридні гілки обміну ключами [4], що збільшує кількість станів і, відповідно, ризик неповноти автомата. Для інфраструктури, побудованої на мові OCaml (унікернели MirageOS, блокчейн Tezos, апаратні модулі безпеки NetHSM), базовою реалізацією TLS є бібліотека ocaml-tls [5], яка вже забезпечує безпеку пам'яті та явну обробку помилок як значень типу Result, проте кодує автомат рукостискання звичайним варіантним типом із узагальнювальною гілкою зіставлення, що приховує від компілятора неповноту автомата.

Дана робота є продовженням досліджень автора щодо застосування системи типів OCaml для забезпечення інваріантів криптографічних інтерфейсів на етапі компіляції [6, 7] і присвячена питанню, чи можна засобами узагальнених алгебраїчних типів даних (GADT), фантомних типів та системи модулів статично виключити зазначені класи помилок у чистому OCaml, не погіршивши продуктивність наявних реалізацій.

Аналіз останніх досліджень і публікацій

Найповніший результат щодо коректності автоматів TLS отримано у проєкті miTLS/Everest, де еталонні реалізації TLS 1.3 верифіковано у мові F* [8, 9]. Спорідненим є формально верифікований криптопровайдер

EverCrypt [10] та бібліотека libcrux [11]. Ці роботи гарантують функціональну коректність і безпеку протоколу, проте вимагають окремої мови доведень і не інтегруються безпосередньо у продукційні реалізації на OCaml. Для протоколів Noise повну автоматизовану верифікацію довільних патернів реалізовано в Noise Explorer [12].

Іншим напрямом є кодування протоколу в системі типів мови реалізації. Теоретичним підґрунтям є сесійні типи [13] та концепція typestate. У Haskell і OCaml сесійні типи реалізовано бібліотечно. FuSe [14] для бінарних сесій та ocaml-mpst [15] для багатосторонніх протоколів із глобальними комбінаторами. Обидві бібліотеки змушені перевіряти лінійність каналу динамічно, оскільки OCaml не має лінійних типі. Статична лінійність через модальності унікальності запропоновано в розширенні OxCaml [16]. Підтримка GADT у OCaml описана в [17], а паралелізм і обробники ефектів OCaml 5 в роботах [18, 19].

Аналіз показує невирішені частини проблеми, такі як відсутність реалізації автомата рукостискання TLS 1.3, у якій відношення переходів є типом, а обов'язковість криптографічних перевірок передумовою конструкторів станів, не досліджено, яка саме частина гарантій сесійних типів досяжна без лінійних типів, відсутність кількісної оцінки накладних витрат такого кодування порівняно з традиційним варіантним автоматом.

Метою роботи є розроблення та експериментальне обґрунтування методу побудови автоматів протоколів автентифікованого узгодження ключів TLS 1.3 та Noise, у якому порушення порядку повідомлень і пропуск обов'язкових перевірок виявляються компілятором. Для досягнення мети розв'язуються наступні задачі. По-перше, надається формалізація автомату рукостискання як індексовану сім'ю типів. По-друге, розроблено типізовані реалізації клієнтського рукостискання TLS 1.3 та протоколу Noise. По-третє, визначено межу між статичними та динамічними перевірками, а також надана експериментальна оцінка повноти статичних гарантій та накладних витрат. У висновках також окреслено шлях інтеграції в бібліотеку ocaml-tls.

Формальна модель типізованого автомата

Нехай Q – скінченна множина станів рукостискання, Σ – множина типів повідомлень, а відношення переходів задано як

$$\delta \subseteq Q \times \Sigma \times Q, \quad (1)$$

У традиційному кодуванні Q подається одним варіантним типом, а δ – тілом функції переходу, завершеної узагальнювальною гілкою. Пропонований метод піднімає Q на рівень типів: кожному стану $q \in Q$ ставиться у відповідність порожній тип-індекс, стан стає індексованою сім'єю 'q state, а δ кодується GADT, у якому конструктор із індексами (q, q') існує тоді й лише тоді, коли $(q, \sigma, q') \in \delta$. Функція переходу отримує тип

$$step : \forall a\ b.\ a\ state \rightarrow (a, b)\ msg \rightarrow (b\ state \times out, fatal)\ result. \quad (2)$$

По суті, (2) є теоремою, яку перевіряє компілятор. Для довільного стану *a* і довільного повідомлення, допустимого в *a*, результатом є стан саме того типу, який приписує δ. Обов'язковість перевірки V на шляху до стану q' кодується доказовим токеном. Функція перевірки повертає значення абстрактного типу V.proof, єдиним виробником якого вона є, а конструктор q' вимагає це значення як поле. Тоді твердження “неможливо перейти у q', не виконавши V” випливає з двох фактів, що перевіряються при компіляції. По-перше, конструктор має поле типу V.proof. По-друге, тип V.proof абстрактний. На відміну від верифікації у F*, доводиться не коректність перевірки, а факт її виклику, проте саме цей факт відсутній у дефектах класу “goto fail”. Вхід з мережі не типізований, тому обов'язково

існує функція-класифікатор, яка зі стану та розпарсеного повідомлення виготовляє типізоване повідомлення з екзистенціально прихованим цільовим індексом: classify : 'a state → handshake → ('a incoming, fatal) result, де 'a incoming = Incoming : ('a,'b) msg → 'a incoming. Це єдине місце, де допустима узагальнювальна гілка, і єдина точка, яку потрібно аудитувати на предмет прийняття зайвого. Класифікатор є чистою функцією і не змінює стан; інакше він став би другою функцією переходу, і межа довіри роздвоїлася б.

Типізований автомат рукостискання TLS 1.3

Реалізовано повне клієнтське рукостискання TLS 1.3 за RFC 8446 без PSK, 0-RTT та HelloRetryRequest (їх додавання розглянуто далі). Криптографічні примітиви абстраговано параметром функтора Make (C : CRYPTO) з мінімальною сигнатурою (хешування, HKDF-Extract/Expand-Label, HMAC, порівняння за сталий час, ECDH, перевірка підпису), що дозволяє підставляти mirage-crypto у продукції та інструментовані реалізації в експериментах. Структуру автомата показано на рис. 1, а ключові визначення у лістингу 1 програми.

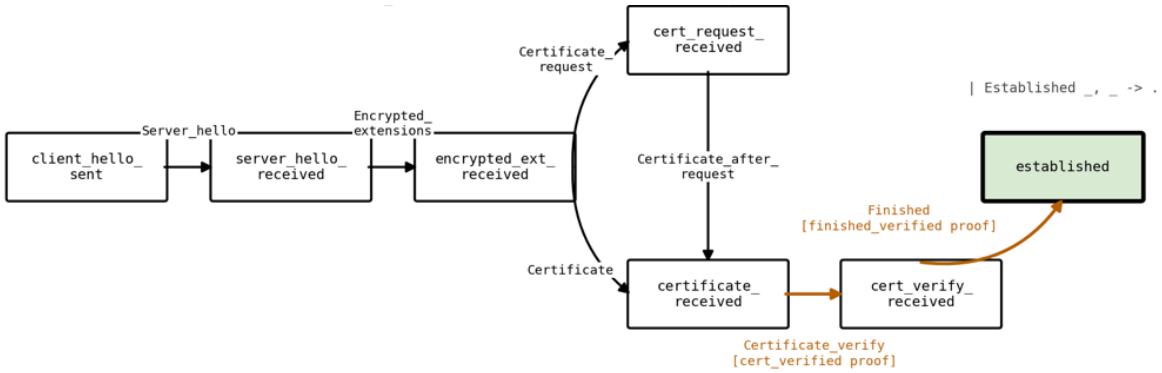


Рис. 1. Типізований автомат рукостискання

```
type client_hello_sent = |   type server_hello_received = |   (* ... *)
type established = |          (* порожні типи-індекси станів *)
module Secret : sig
  type early = |  type handshake = |  type master = |
  type 'level t                      (* фантомний рівень key schedule *)
  val handshake : early t -> shared:string -> handshake t
  val master    : handshake t -> master t
end
module Verified : sig
  type cert_verified = |  type finished_verified = |
  type 'what proof                   (* абстрактний, єдиний виробник - перевірка *)
  val finished : finished_key:string -> transcript_hash:string -> finished
              -> (finished_verified proof, [> `Bad_finished ]) result
end
type (_, _) msg =                    (* публічний GADT = діаграма станів *)
  | Server_hello : server_hello -> (client_hello_sent, server_hello_received) msg
  | Certificate  : certificate  -> (encrypted_extensions_received, certificate_received) m
sg
  | Certificate_verify : certificate_verify
                      -> (certificate_received, certificate_verify_received) msg
  | Finished     : finished -> (certificate_verify_received, established) msg  (* ... *)
val step : 'a state -> raw:string -> ('a, 'b) msg -> ('b state * output list, [> fatal ])
result
```

Лістинг 1. Типізований автомат рукостискання

Індекси станів реалізовано порожніми типами (OCaml ≥ 4.07), а не поліморфними варіантами: порожні типи ін'єктивні, не потребують обмежень [>] та дають короткі повідомлення про помилки. Розклад ключів RFC 8446 (розд. 7.1) отримує фантомний рівень: функція master приймає лише handshake t, тому неможливо вивести трафік-секрети застосунку з early-секрету. Конструктор

Established вимагає двох доказових токенів – перевірки CertificateVerify та Finished; токен сертифіката “протягується” через проміжний стан і не може бути втрачений. Модуль Verified має власну сигнатуру, яка приховує представлення proof (внутрішньо – unit) навіть від решти тіла функтора.

```
let step : type a b. a state -> raw:string -> (a, b) msg
        -> (b state * output list, [> fatal ]) result = fun st ~raw m ->
  match st, m with
  | Client_hello_sent { ephemeral_priv; early; transcript }, Server_hello sh -> ...
  | Certificate_received { ctx; leaf_pubkey; client_auth }, Certificate_verify cv ->
      (match Verified.certificate_verify ~pubkey:leaf_pubkey ~transcript_hash:th cv with
       | Error _ as e -> e
       | Ok cert_ok -> Ok (Certificate_verify_received { ctx; cert_ok; client_auth }, []))
  | Certificate_verify_received { ctx; cert_ok; _ }, Finished fin ->
      (match Verified.finished ~finished_key:fk ~transcript_hash:th_before fin with
       | Error _ as e -> e
       | Ok server_fin_ok ->
           Ok (Established { master; app_client; app_server; cert_ok; server_fin_ok },
               [ Send_finished client_fin ]))
  | Established _, _ -> .   (* refutation clause: порожність перевіряє типізатор *)
```

Лістинг 2. Функція переходу без узагальненої гілки

Функція переходу містить сім конструктивних гілок за кількістю ребер δ і не містить узагальнювальної гілки. Експериментально встановлено, що перевірка вичерпності OCaml не доводить порожність типу (established, 'b) msg самостійно і без останнього рядка видає попередження 8 з контрприкладом.

Протокол Noise: патерн рукостискання як типорівневий список

Специфікація Noise задає рукостискання патерном тобто послідовністю повідомлень, кожне з яких є списком токенів e, s, ee, es, se, ss. Семантика протоколу повністю визначена цим списком, що робить його природним об'єктом для підняття на рівень типів. Токени кодуються GADT token з індексами-порожніми типами, повідомлення типорівневим списком tokens, а патерн як GADT pattern, індексованим парою. Два конструктори I_send та They_send кодують чергування ролей без типорівневих функцій, яких у OCaml немає. Значення xx_initiator і xx_responder записуються звичайним кодом, а їхні типи виводить компілятор.

Типи API є гарантіями: write_message існує лише для стану, де черга власна, і повертає стан, де черга співрозмовника; split приймає лише стан із вичерпаним патерном. Тип error містить виключно криптографічні відмови (Decrypt_failed, Truncated, Missing_remote_key) – варіанта “не своя черга” немає, бо він недосяжний.

Експериментальні результати

Усі артефакти скомпільовано OCaml 4.14.1 з прапорцями -w +a -principal. Код не використовує конструкцій, що змінилися в OCaml 5.x.

Оцінювалися чотири гіпотези: повнота статистичних гарантій, накладні витрати, відсутність нових секрето-залежних розгалужень та супровідність.

```
type (_, _) pattern =
  | Done      : ('turn, nil) pattern
  | I_send    : 't tokens * (them, 'rest) pattern -> (me,   ('t, 'rest) cons) pattern
  | They_send : 't tokens * (me,   'rest) pattern -> (them, ('t, 'rest) cons) pattern
val write_message : (me, ('t,'rest) cons) hs -> payload:string
                 -> ((them,'rest) hs * string, error) result
val read_message  : (them, ('t,'rest) cons) hs -> string
                 -> ((me,'rest) hs * string, error) result
val split         : ('turn, nil) hs -> cipher_state * cipher_state
```

Лістинг 3. Типорівневий патерн Noise

Повнота статистичних гарантій

Згенеровано всі |Q|×|Σ| = 49 програм вигляду step st (Ctor payload), де st має тип q state, і кожну подано компілятору. Результати наведено в табл. 1: компілятор прийняв рівно 7 програм, що збігаються з множиною ребер δ, і відхилив 42. Аналогічна матриця для Noise XX (4 стани × 3 операції) дала 4 прийнятих із 12 (табл. 2). Таким чином, статична частина гарантій є повною відносно δ і не потребує фазингу послідовностей повідомлень, як у FlexTLS [1].

Таблиця 1. Результати компіляції програм step для всіх пар «стан – повідомлення» автомата TLS 1.3

Стан \ повідомлення	**SH**	**EE**	**CR**	**CT**	**CTr**	**CV**	**FIN**
client_hello_sent	+	–	–	–	–	–	–
server_hello_received	–	+	–	–	–	–	–
encrypted_extensions_received	–	–	+	+	–	–	–
certificate_request_received	–	–	–	–	+	–	–
certificate_received	–	–	–	–	–	+	–
certificate_verify_received	–	–	–	–	–	–	+
established	–	–	–	–	–	–	–

Позначення: "+" – програма компілюється (перехід допустимий), "–" – помилка типізації. SH – Server_hello, EE – Encrypted_extensions, CR – Certificate_request, CT – Certificate, CTr – Certificate_after_request, CV – Certificate_verify, FIN – Finished.

Таблиця 2. Результати компіляції операцій API для станів ініціатора патерну Noise XX

Стан ініціатора XX	**write_message**	**read_message**	**split**
до повідомлення 1	+	–	–
до повідомлення 2	–	+	–
до повідомлення 3	+	–	–
патерн вичерпано	–	–	+

Окремо перевірено три характерні атаки: прийняття Finished одразу після ClientHello (Early Finished) відхиляється як невідповідність індексів (certificate_verify_received, established) msg та (client_hello_sent, 'a) msg; фабрикація токена значенням () відхиляється як невідповідність unit і finished_verified proof; запит ключів застосунку з проміжного стану відхиляється як

невідповідність client_hello_sent state і established state. Динамічні перевірки залишаються лише на межі довіри: у прогоні з іграшковою криптографією чесне рукостискання завершується станом established, підроблений Finished відхиляється з кодом Bad_finished, а Finished у неправильному стані з кодом Unexpected_message ще у класифікаторі.

Накладні витрати. Для ізоляції вартості автомата від вартості криптографії використано заглушку CRYPTO з операціями O(1) та нетипізований еталон – реалізацію тієї самої логіки плоским варіантним типом із узагальнювальною гілкою, як в ocaml-tls. Вимірювалося повне п'ятикрокове рукостискання у native-коді (ocamlopt 4.14.1, x86-64, одне ядро), медіана 11 серій по $3 \cdot 10^6$ рукостискань; алокації – за лічильником Gc.minor_words. Результати подано в табл. 3 і на рис. 2.

Таблиця 3. Накладні витрати типізованого автомата порівняно з нетипізованим еталоном

Показник	Нетипізований (wildcard)	Типізований (GADT)	Різниця
Час, нс на рукостискання	291–309	341–368	+33…+63 нс (+10…+22 %)
Час, нс на крок автомата	58–62	68–74	≈ +10 нс
Алокації, слів minor heap на рукостискання	195	268	+73 слова (584 байти)
Частка від рукостискання ECDHE-P256 (≈130 мкс)	0,2 %	0,3 %	< 0,05 %

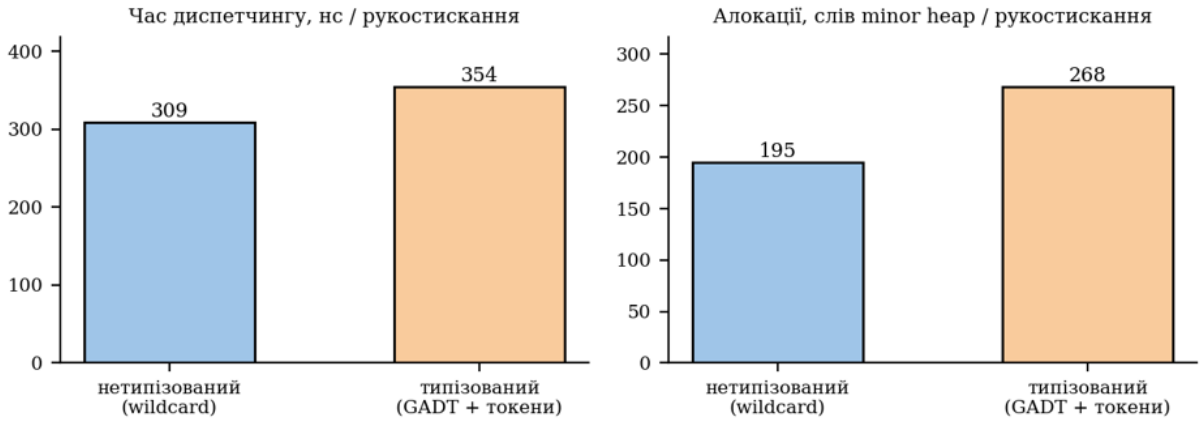


Рис. 2. Порівняння накладних витрат типізованого автомата відносно нетипізованого еталону

Джерела додаткових витрат ідентифіковано. Це екзистенціальні обгортки Incoming і Packed на межі з нетипізованим циклом (по одному блоку на крок), два поля-токени в Established (два слова на рукостискання) та вкладений запис контексту. Індекси станів і фантомні рівні стираються компілятором і не мають вартості. За абсолютною величиною надлишок становить десятки наносекунд, тоді як одне рукостискання TLS 1.3 з ECDHE на P-256 у mirage-crypto триває близько 130 мкс [20], тобто вплив на пропускну здатність лежить нижче роздільної здатності вимірювань і не є бар'єром для інтеграції.

Побічні канали. Автомат не вводить розгалужень, залежних від секретів. Усі зіставлення виконуються за типом повідомлення та станом, які є публічними.

Єдина секрето-залежна операція – порівняння у Verified.finished, делеговане функції сталого часу. Для підтвердження на інтегрованій реалізації передбачено методику dudect [21] з двома класами входів фіксованої довжини та порогом $|t| < 4{,}5$ після 10^6 вимірювань.

Супровідність. Тестування типів показало вилучення гілки step породжує попередження 8, додавання конструктора в msg – попередження 8 або помилку в запереченні застережень, а вилучення виклику Verified.* – помилку типізації. Усі три властивості придатні для автоматизації в CI.

Межі методу та інтеграція в ocaml-tls. Класи помилок, які виключає та не виключає метод, зведено в табл. 4. Принциповою межею є лінійність. Після переходу старе значення стану залишається доступним, і OCaml 5.x не забороняє його повторного використання.

Розглянуто чотири варіанти:

- динамічна перевірка спожитості за прикладом FuSe та ocaml-mpst [14, 15];
- API у стилі передавання продовжень (не дає гарантії);
- дисципліна модуля, за якої конструктори станів бачить лише функція переходу, а цикл рушія є єдиним і коротким;
- модальності унікальності OxCaml [16], які роблять повторне використання помилкою компіляції. Для інтеграції в ocaml-tls обрано третій варіант, оскільки цикл Engine.handle_tls у бібліотеці і так єдиний, а четвертий визначено як перспективний.

Таблиця 4. Класи помилок керування станом, які виключає типізований автомат, порівняно з ocaml-tls

Клас помилки	Приклад	ocaml-tls (поточний)	Типізований автомат
Прийняття повідомлення у неправильному стані	Early Finished, пропуск ServerKeyExchange [1]	динамічно (fail-closed)	динамічно лише в класифікаторі; далі статично
Неповнота автомата після рефакторингу	додано стан або повідомлення	непомітна (wildcard)	попередження 8 / заперечення
Пропущена криптоперевірка на шляху до стану	goto fail (CVE-2014-1266)	властивість тіла функції	статично (доказовий токен)
Секрет не того рівня key schedule	hs-секрет замість app-секрету	властивість тіла функції	статично (фантомний рівень)
Ключі застосунку до завершення рукостискання	помилка API	динамічно	статично
Дія не в свою чергу (Noise)	write замість read	–	статично
Повторне використання старого стану	подвійний Finished	не виключено	не виключено (потрібні лінійні типи)
Помилка парсера, обхід через Obj.magic	_	поза межами	поза межами

План інтеграції не передбачає переписування ocaml-tls. Замінюються лише ядра Handshake_client13 та Handshake_server13 із збереженням Engine, серіалізації алертів і всіх транспортів (Lwt, Eio, Miou, MirageOS).

На першому етапі варіант machina отримує конструктор Client13_typed of packed, а транскрипт реалізується

інкрементальним контекстом хешування замість списку. Гілки протоколу додаються як додаткові індекси. HelloRetryRequest додається лічильником повторів (once/twice), за якого другий повтор не має конструктора, а PSK-режим додається індексом способу автентифікації (cert_auth/psk_auth), який обирає класифікатор за розширенням pre_shared_key повідомлення ServerHello. Клієнтська автентифікація – підняттям поля client_auth у фантом. Для TLS 1.2, де композитний автомат найбільший, перспективним є індексування не станом, а множиною ще не отриманих обов'язкових повідомлень за аналогією з Noise, що стискає кількість конструкторів.

Практичні труднощі виведення типів локалізовано у визначенні автомата: усі функції над GADT потребують локально абстрактних типів (type a b.).

Висновки

1. Запропоновано метод побудови автоматів рукостискання криптографічних протоколів у мові OCaml, за якого відношення переходів є узагальненим алгебраїчним типом даних, функція переходу тотальна без узагальнювальної гілки, а обов'язковість криптографічних перевірок є передумовою конструкторів станів через доказові токени. Порушення порядку повідомлень поза єдиною функцією-класифікатором і пропуск перевірок класу “goto fail” стають помилками компіляції.

2. Експериментально підтверджено повноту статичних гарантій: з 49 можливих пар. Фантомні рівні розкладу ключів додатково виключають плутанину секретів без витрат на виконання.

3. Накладні витрати кодування становлять 33–63 нс та 73 слова алокацій на рукостискання (10–22 % часу диспетчингу автомата за заглушеної криптографії), що менше ніж 0,05 % тривалості реального рукостискання з ECDHE і не є перешкодою для інтеграції в ocaml-tls.

4. Встановлено межу методу: лінійність стану не забезпечується у стандартному OCaml 5.x і компенсується дисципліною модуля рушія. Статичне рішення очікується від модальностей унікальності OxCaml. Виявлено, що перевірка вичерпності GADT потребує явних заперечень застережень для поглинальних станів, які слід трактувати як задокументовані інваріанти.

Перспективи подальших досліджень. Цікавим напрямом є інтеграція методу в ocaml-tls з вимірюванням на реальній криптографії та dudect-аналізом. Наступною темою також може стати розширення індексації на HelloRetryRequest, PSK та гібридний постквантовий обмін ключами. Потребує зусиль перевірка гіпотези про стискання композитного автомата TLS 1.2 індексуванням множиною неотриманих повідомлень. Практичний інтерес також має перенесення дизайну на OxCaml для статичної лінійності.

Література

1. B. Beurdouche et al., "A Messy State of the Union: Taming the Composite State Machines of TLS," in Proc. IEEE Symp. Security and Privacy (S&P), San Jose, CA, USA, 2015, pp. 535–552, doi: 10.1109/SP.2015.39.

2. E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," IETF RFC 8446, Aug. 2018, doi: 10.17487/RFC8446.

3. T. Perrin, "The Noise Protocol Framework," Revision 34, 2018. [Online]. Available: https://noiseprotocol.org/noise.html

4. D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," IETF Internet-Draft draft-ietf-tls-hybrid-design, 2025.

5. H. Mehnert and D. Kaloper-Meršinjak, "ocaml-tls: TLS in pure OCaml," 2026. [Online]. Available: https://github.com/mirleft/ocaml-tls

6. Y. Kotukh, A. Honcharuk, M. Korobchynskyi, I. Kiris, M. Rudenko, and V. Voitko, "Type Safe OCaml Bindings for the

Open Quantum Safe Post-Quantum Cryptography Library," in Proc. IEEE Int. Conf., 2026, accepted for publication.

7. Y. Kotukh, G. Khalimov, I. Dzhura, and H. Hivrenko, "Application of the LINE encryption scheme in the key encapsulation mechanism for the authentication protocol in 5G networks," Radiotekhnika, no. 219, pp. 36–45, 2024, doi: 10.30837/rt.2024.4.219.04.

8. K. Bhargavan, B. Blanchet, and N. Kobeissi, "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate," in Proc. IEEE Symp. Security and Privacy (S&P), 2017, pp. 483–502, doi: 10.1109/SP.2017.26.

9. A. Delignat-Lavaud et al., "Implementing and Proving the TLS 1.3 Record Layer," in Proc. IEEE Symp. Security and Privacy (S&P), 2017, pp. 463–482, doi: 10.1109/SP.2017.58.

10. J. Protzenko et al., "EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider," in Proc. IEEE Symp. Security and Privacy (S&P), 2020, pp. 983–1002, doi: 10.1109/SP40000.2020.00114.

11. Cryspen, "libcrux: Formally verified cryptography in Rust," 2025. [Online]. Available: https://github.com/cryspen/libcrux

12. N. Kobeissi, G. Nicolas, and K. Bhargavan, "Noise Explorer: Fully Automated Modeling and Verification for Arbitrary Noise Protocols," in Proc. IEEE European Symp. Security and Privacy (EuroS&P), 2019, pp. 356–370, doi: 10.1109/EuroSP.2019.00034.

13. S. J. Gay and M. Hole, "Subtyping for session types in the pi calculus," Acta Informatica, vol. 42, no. 2–3, pp. 191–225, 2005, doi: 10.1007/s00236-005-0177-z.

14. L. Padovani, "A simple library implementation of binary sessions," Journal of Functional Programming, vol. 27, e4, 2017, doi: 10.1017/S0956796816000289.

15. K. Imai, R. Neykova, N. Yoshida, and K. Yuen, "Multiparty Session Programming with Global Protocol Combinators," in Proc. 34th European Conf. Object-Oriented Programming (ECOOP), LIPIcs vol. 166, 2020, pp. 9:1–9:30, doi: 10.4230/LIPIcs.ECOOP.2020.9.

16. A. Lorenzen, L. White, S. Dolan, R. A. Eisenberg, and S. Lindley, "Oxidizing OCaml with Modal Memory Management," Proc. ACM Program. Lang., vol. 8, no. ICFP, art. 253, 2024, doi: 10.1145/3674642.

17. J. Garrigue and D. Rémy, "Ambivalent Types for Principal Type Inference with GADTs," in Proc. Asian Symp. Programming Languages and Systems (APLAS), LNCS vol. 8301, 2013, pp. 257–272, doi: 10.1007/978-3-319-03542-0_19.

18. K. C. Sivaramakrishnan, S. Dolan, L. White, S. Jaffer, T. Kelly, A. Sahoo, S. Parimala, A. Dhiman, and A. Madhavapeddy, "Retrofitting Parallelism onto OCaml," Proc. ACM Program. Lang., vol. 4, no. ICFP, art. 113, 2020, doi: 10.1145/3408995.

19. K. C. Sivaramakrishnan, S. Dolan, L. White, T. Kelly, S. Jaffer, and A. Madhavapeddy, "Retrofitting Effect Handlers onto OCaml," in Proc. 42nd ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI), 2021, pp. 206–221, doi: 10.1145/3453483.3454039.

20. MirageOS Project, "mirage-crypto: Cryptographic primitives for OCaml – CHANGES," 2026. [Online]. Available: https://github.com/mirage/mirage-crypto/blob/main/CHANGES.md

21. O. Reparaz, J. Balasch, and I. Verbauwhede, "Dude, is my code constant time?" in Proc. Design, Automation & Test in Europe Conf. (DATE), 2017, pp. 1697–1702, doi: 10.23919/DATE.2017.7927267.

Котух Є.В., Шкляр О.І.

ЗАСТОСУВАННЯ ТИПОБЕЗПЕЧНИХ АВТОМАТІВ У МОВІ OCAML ДЛЯ АВТЕНТИФІКОВАНОГО УЗГОДЖЕННЯ КЛЮЧІВ

Значна частка вразливостей протоколів захищеного зв'язку, зокрема у вузлах TLS-термінації балансувальників навантаження хмарної інфраструктури, зумовлена помилками керування станом рукостискання. Запропоновано метод побудови автоматів протоколів TLS 1.3 та Noise мовою OCaml, у якому відношення переходів закодовано узагальненим алгебраїчним типом даних (GADT), а обов'язковість криптографічних перевірок – доказовими токенами, тому порушення порядку повідомлень і пропуск перевірок виявляє компілятор. Експериментально встановлено, що з 49 пар «стан – повідомлення» TLS 1.3 компілятор приймає рівно 7, які відповідають RFC 8446, а накладні витрати не перевищують 0,05 % тривалості рукостискання і не знижують пропускної здатності вузлів TLS-термінації. Окреслено межі методу та план інтеграції в бібліотеку ocaml-tls для унікернелів MirageOS.

Ключові слова: *OCaml, узагальнені алгебраїчні типи даних (GADT), типізований автомат станів, TLS 1.3, протокол Noise, автентифіковане узгодження ключів, TLS-термінація, балансування навантаження, унікернели, MirageOS.*

Kotukh Ye., Shklyar

O. APPLICATION OF TYPE-SAFE STATE MACHINES IN OCAML FOR AUTHENTICATED KEY EXCHANGE

A significant share of vulnerabilities in secure communication protocols, including those in the TLS termination nodes of cloud infrastructure load balancers, stems from handshake state management errors. A method is proposed for constructing TLS 1.3 and Noise state machines in OCaml in which the transition relation is encoded as a generalized algebraic data type (GADT) and mandatory cryptographic checks are enforced by proof tokens, so message ordering violations and omitted checks are detected by the compiler. Experiments show that out of 49 TLS 1.3 state–message pairs the compiler accepts exactly 7, matching RFC 8446, while the overhead does not exceed 0.05 % of handshake time and does not reduce the throughput of TLS termination nodes. The limitations of the method and a plan for integration into the ocaml-tls library for MirageOS unikernels are outlined.

Keywords: *OCaml, generalized algebraic data types (GADT), typed state machine, TLS 1.3, Noise protocol, authenticated key exchange, TLS termination, load balancing, unikernels, MirageOS*.

Стаття подана до редакції: 13/09/2026
Стаття прийнята до опублікування:15/09/2026
Стаття опублікована: 10/11/2026

Стаття поширюється на умовах ліцензії CC BY 4.0