Whitepaper: Kiến trúc mật mã học TJGuard

Tác giả: Vưu Trường Nhật Thanh

Ngày phát hành: 01/04/2026

Phiên bản: 1.0

Tóm tắt

TJGuard là một ứng dụng quản lý mật khẩu theo mô hình zero-knowledge — máy chủ không bao giờ nhận, lưu trữ, hay xử lý dữ liệu người dùng ở dạng văn bản thô. Toàn bộ quá trình mã hóa và giải mã đều diễn ra hoàn toàn trên thiết bị của người dùng (client-side), sử dụng các thuật toán mật mã học hiện đại đã được kiểm chứng bởi cộng đồng bảo mật quốc tế. Tài liệu này mô tả chi tiết các thuật toán, tham số kỹ thuật, và luồng hoạt động của hệ thống mật mã trong TJGuard nhằm giúp người dùng và cộng đồng kỹ thuật đánh giá độc lập mức độ bảo mật của sản phẩm.

1. Triết lý thiết kế

1.1. Zero-Knowledge là gì?

Trong mô hình zero-knowledge, nhà cung cấp dịch vụ (TJGuard) về mặt kỹ thuật không có khả năng đọc dữ liệu của người dùng, ngay cả khi muốn. Điều này khác với các mô hình truyền thống nơi nhà cung cấp giữ khóa mã hóa và chỉ "hứa" không đọc dữ liệu.

Với TJGuard:

- Master password không bao giờ được gửi lên máy chủ.
- Khóa mã hóa chỉ tồn tại trong RAM của trình duyệt người dùng trong thời gian phiên làm việc.
- Máy chủ chỉ lưu trữ các blob đã mã hóa không thể đọc được nếu không có master password.
- Khi người dùng đăng xuất hoặc bất hoạt quá 10 phút, tất cả khóa trong RAM được xóa hoàn toàn.

1.2. Nguyên tắc cốt lõi

Nguyên tắc	Hiện thực hóa trong TJGuard
Bảo mật ngay cả khi máy chủ bị xâm phạm	Chỉ lưu ciphertext, không có plaintext hay khóa
Thuật toán chuẩn quốc tế	Argon2id, XChaCha20-Poly1305, Ed25519
Không tin tưởng mù quáng	Có thể kiểm toán kỹ thuật độc lập
Bảo vệ tối thiểu bề mặt tấn công	Khóa chỉ sống trong RAM, tự xóa sau 10 phút

2. Các thành phần mật mã học

TJGuard sử dụng bốn nhóm thuật toán mật mã, mỗi nhóm phục vụ một mục đích cụ thể:

2.1. Hàm dẫn xuất khóa từ mật khẩu — Argon2id

Mục đích: Biến master password (một chuỗi ký tự người nhớ được) thành một khóa mã hóa 256-bit (32 byte) có độ entropy cao.

Vấn đề cần giải quyết: Mật khẩu do người dùng tạo thường có entropy thấp và dễ bị tấn công brute-force. Hàm KDF (Key Derivation Function) được thiết kế để làm cho mỗi lần thử đoán mật khẩu cực kỳ tốn kém về mặt tính toán.

Thuật toán: Argon2id — phiên bản lai giữa Argon2i (kháng side-channel) và Argon2d (kháng GPU attack), đạt giải nhất cuộc thi Password Hashing Competition (PHC) năm 2015, được chuẩn hóa bởi IETF tại RFC 9106 [1].

Tham số TJGuard:

```
Algorithm:      Argon2id

Memory cost:    128 MB (131,072 KB)
```

```
Time cost:      3 iterations

Parallelism:    1 thread

Output length:  32 bytes (256 bits)

Salt:           16 bytes random (stored on server, not secret)
```

Tại sao chọn các tham số này?

- 128 MB memory cost: Buộc kẻ tấn công phải dùng 128 MB RAM cho mỗi lần thử, làm cho các cuộc tấn công song song bằng GPU/ASIC trở nên không khả thi về kinh tế. Với một GPU RTX 4090 có 24 GB VRAM, kẻ tấn công chỉ chạy được ~187 thử nghiệm song song thay vì hàng triệu.
- 3 iterations: Tăng thêm thời gian tính toán trên CPU, không ảnh hưởng đáng kể đến trải nghiệm người dùng (< 1 giây) nhưng tăng chi phí tấn công.
- Salt ngẫu nhiên: Mỗi người dùng có salt riêng, ngăn chặn tấn công rainbow table và đảm bảo hai người dùng có cùng mật khẩu vẫn ra khóa khác nhau.

Implementation: Thư viện hash-wasm (WebAssembly) — chạy trực tiếp trong trình duyệt, không cần gửi mật khẩu lên server.

2.2. Mã hóa đối xứng xác thực — XChaCha20-Poly1305

Mục đích: Mã hóa dữ liệu vault với tính năng Authenticated Encryption with Associated Data (AEAD) — đảm bảo dữ liệu vừa bí mật vừa không bị giả mạo.

Thuật toán: XChaCha20-Poly1305, gồm hai phần:

- XChaCha20: Thuật toán mã hóa stream cipher với nonce 192-bit (24 byte). Là phiên bản mở rộng của ChaCha20 (RFC 8439 [2]), thiết kế bởi Daniel

J. Bernstein. Nonce lớn hơn giúp tạo ngẫu nhiên an toàn hơn mà không lo nguy cơ nonce tái sử dụng.

- Poly1305: MAC (Message Authentication Code) 128-bit, đảm bảo tính toàn vẹn — bất kỳ sự thay đổi nào vào ciphertext đều bị phát hiện khi giải mã.

Tham số TJGuard:

```
Algorithm:      XChaCha20-Poly1305

Key size:       32 bytes (256 bits)

Nonce size:     24 bytes (192 bits) — random per encryption

Tag size:       16 bytes (128 bits) MAC

AAD:            Vault owner's email (Additional Authenticated
Data)
```

Tại sao XChaCha20-Poly1305 thay vì AES-GCM?

Tiêu chí	**XChaCha20-Poly1305**	**AES-GCM**
Kích thước nonce	24 byte (an toàn random hơn)	12 byte (cần cẩn thận hơn)
Phụ thuộc phần cứng	Không cần AES-NI	Cần hỗ trợ phần cứng
Timing attacks	Constant-time by design	Phụ thuộc implementation
WebAssembly performance	Rất tốt	Phụ thuộc platform
Tiêu chuẩn	RFC 8439 [2] + XSalsa20 extension	NIST SP 800-38D

Additional Authenticated Data (AAD): TJGuard dùng email người dùng làm AAD. Điều này đảm bảo rằng một ciphertext chỉ có thể giải mã thành công trong ngữ cảnh của đúng tài khoản — ngăn chặn tấn công "copy ciphertext sang tài khoản khác".

Implementation: Thư viện @stablelib/xchacha20poly1305 — thuần JavaScript/TypeScript, đã được kiểm toán bảo mật.

2.3. Dẫn xuất khóa theo ngữ cảnh — HKDF-SHA256

Mục đích: Từ một khóa gốc (Account Key), dẫn xuất ra các khóa con độc lập cho từng mục trong vault, đảm bảo việc lộ khóa của một mục không ảnh hưởng đến các mục khác.

Thuật toán: HKDF (HMAC-based Key Derivation Function) với SHA-256, được chuẩn hóa tại RFC 5869 [3] và NIST SP 800-56C [4].

Tham số TJGuard:

```
Algorithm:      HKDF-SHA256

IKM (Input):    Account Key (32 bytes)

Salt:           16 bytes random (stored with vault item)

Info:           "tjguard:item:v1" (UTF-8 domain separator)

Output length:  32 bytes (256 bits)
```

Tại sao dùng HKDF?

Không thể dùng trực tiếp Account Key để mã hóa nhiều mục vì:

- Key reuse risk: Dùng cùng khóa + cùng nonce = thảm họa bảo mật.
- Forward secrecy cục bộ: Nếu khóa một mục bị lộ, các mục khác vẫn an toàn.

- Domain separation: Tham số info = "tjguard:item:v1" đảm bảo các khóa dẫn xuất cho các mục đích khác nhau không trùng nhau.

Implementation: Web Crypto API của trình duyệt — API mật mã học cấp thấp được tích hợp sẵn, không phụ thuộc thư viện bên thứ ba.

2.4. Mã hóa bất đối xứng — Ed25519 / NaCl Box

Mục đích: Cho phép người dùng chia sẻ vault item với người khác mà không cần tiết lộ khóa mã hóa, thông qua cơ chế public-key cryptography.

Thuật toán: NaCl box (Networking and Cryptography Library), dựa trên:

- X25519 (RFC 7748 [5]): Elliptic Curve Diffie-Hellman (ECDH) để thỏa thuận khóa chung.
- XSalsa20-Poly1305: AEAD để mã hóa dữ liệu sau khi có shared secret.

Tham số TJGuard:

```
Library:        TweetNaCl (JavaScript port of NaCl)

Key pair:       nacl.box.keyPair() — Ed25519-compatible

Secret key:     32 bytes

Public key:     32 bytes (stored publicly on server)

Nonce:          24 bytes random per seal

Ephemeral key:  Freshly generated per share
```

Luồng chia sẻ vault item:

```
Người chia sẻ (Owner):

1. Lấy public key của người nhận từ server

2. Tạo ephemeral keypair mới

3. ECDH(ephemeral_secret, recipient_public) → shared_secret

4. Mã hóa: nacl.box(itemKey, nonce, recipient_public,
ephemeral_secret)

5. Gửi lên server: [ephPub(32B) || nonce(24B) || ciphertext]



Người nhận (Recipient):

1. Lấy wrapped_item_key từ server

2. Giải nén: ephPub + nonce + ciphertext

3. ECDH(recipient_secret, ephPub) → shared_secret

4. Giải mã: nacl.box.open(ciphertext, nonce, ephPub,
recipient_secret) → itemKey

5. Dùng itemKey + XChaCha20-Poly1305 để giải mã nội dung vault
item
```

Bảo vệ private key: Private key của người dùng không bao giờ lưu dạng thô. Nó được mã hóa bằng Account Key (XChaCha20-Poly1305) trước khi gửi lên server:

```
enc_private_wrapped = XChaCha20-Poly1305.seal(privateKey,
accountKey)
```

2.5. Sinh số ngẫu nhiên an toàn

Tất cả các giá trị ngẫu nhiên trong TJGuard (salt, nonce, keypair) đều được sinh bởi Web Crypto API của trình duyệt thông qua `crypto.getRandomValues()`. Đây là CSPRNG (Cryptographically Secure Pseudo-Random Number Generator) được chuẩn hóa theo NIST SP 800-90A [6], sử dụng entropy từ hệ điều hành (tương đương /dev/urandom trên Linux). Không có ngẫu nhiên "giả" hay seed cố định.

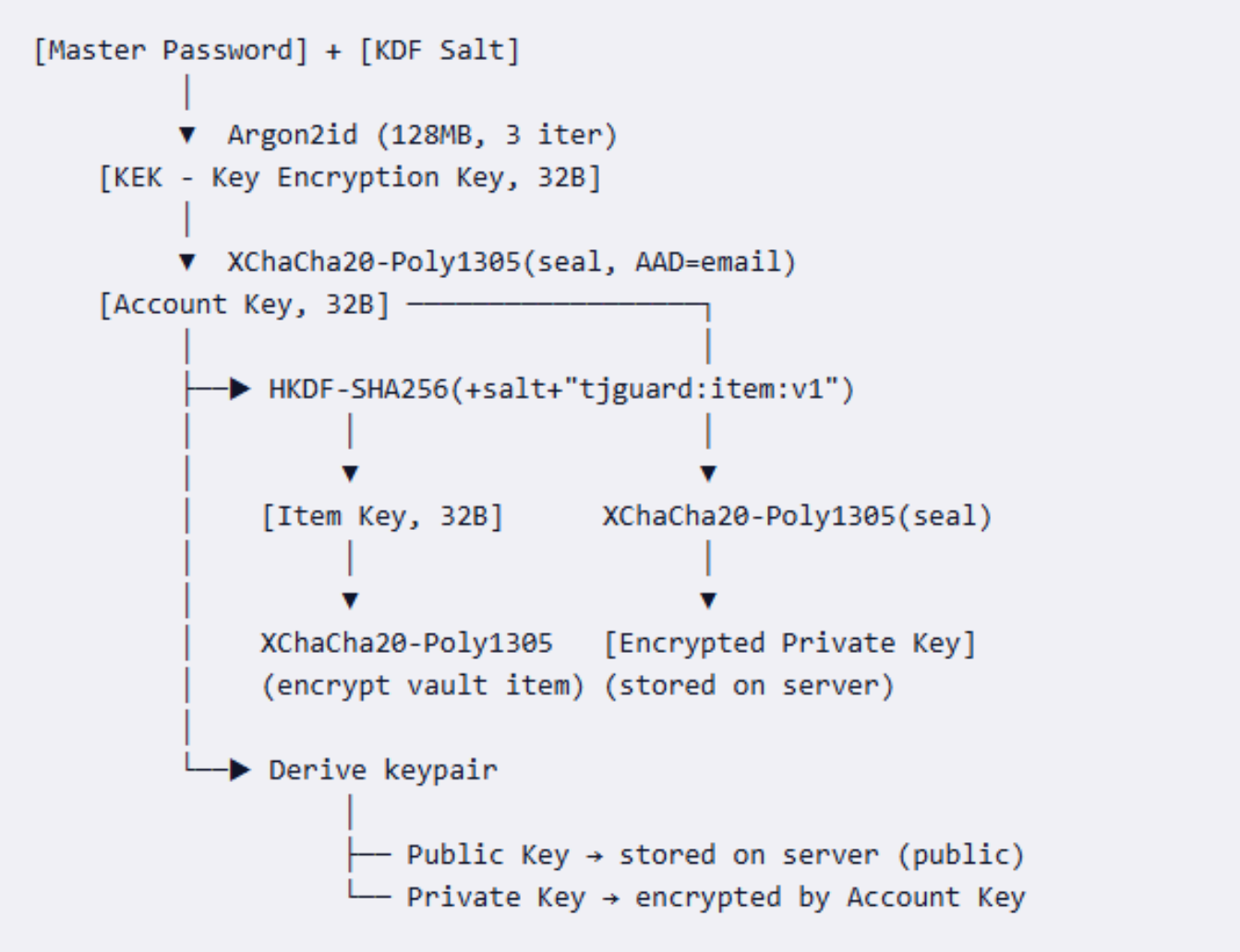
2.6. Xác thực hai yếu tố — TOTP

Thuật toán: TOTP (Time-based One-Time Password) theo RFC 6238 [7], tương thích với Google Authenticator, Authy, v.v.

```
TOTP = HOTP(secret, floor(timestamp / 30))

HOTP = HMAC-SHA1(secret, counter)

Output: 6 digits

Window: ±30 seconds (1 step) for clock drift tolerance
```

Secret TOTP được lưu mã hóa trên server và chỉ dùng để xác thực — không liên quan đến khóa mã hóa vault.

3. Kiến trúc phân cấp khóa



Ý nghĩa kiến trúc này:

- Thay đổi master password không cần mã hóa lại toàn bộ vault — chỉ cần wrap lại Account Key với KEK mới.
- Recovery Key = bản sao Account Key được tải xuống an toàn — cho phép khôi phục vault mà không cần master password.
- Chia sẻ vault dùng Item Key wrap bằng public key người nhận — không tiết lộ Account Key.
- Xóa quyền chia sẻ yêu cầu rotate Item Key mới — người đã bị thu hồi không thể giải mã bản mới.

4. Mô hình bảo mật và phân tích mối đe dọa

4.1. Kịch bản: Máy chủ TJGuard bị xâm phạm

Dữ liệu trên server	Kẻ tấn công thu được	Đọc vault?
kdf_salt	Salt của Argon2id	Không — vẫn cần master password
ak_wrapped	Account Key đã mã hóa	Không — cần KEK
enc_private_wrapped	Private key đã mã hóa	Không — cần Account Key
ciphertext	Vault item đã mã hóa	Không — cần Item Key
enc_public_key	Public key người dùng	Public key không bí mật

"Ngay cả khi database bị dump toàn bộ, kẻ tấn công không thể đọc bất kỳ vault item nào mà không biết master password của từng người dùng."

4.2. Kịch bản: Tấn công brute-force master password

Với Argon2id (128 MB, 3 iterations) theo RFC 9106 [1] :

- Thời gian tính toán mỗi lần thử: ~300-800ms trên phần cứng thông thường
- Với GPU RTX 4090 (24 GB VRAM): ~187 luồng song song, ~600 lần thử/phút (ước tính)
- Với master password 12 ký tự ngẫu nhiên (chữ hoa + thường + số + ký tự đặc biệt): không gian tìm kiếm ≈ 2^78
- Thời gian brute-force toàn bộ không gian: vượt xa thời gian hữu ích

Lưu ý: các ước tính trên giả định master password được sinh ngẫu nhiên. Password do người tạo thường có entropy thấp hơn nhiều — bạn nên dùng bộ sinh password của TJGuard.

4.3. Kịch bản: Nonce tái sử dụng

Với XChaCha20-Poly1305 (RFC 8439 [2]), nonce 24 byte (192 bit):

- Xác suất collision với 2^64 lần mã hóa: ~10^-18 (negligible)
- Mỗi vault item dùng nonce ngẫu nhiên mới từ `crypto.getRandomValues()`
- Rủi ro thực tế: cực kỳ thấp ngay cả với hàng triệu vault item

4.4. Kịch bản: Man-in-the-Middle (MITM)

- Toàn bộ giao tiếp qua HTTPS (TLS 1.2/1.3)
- Dù MITM có giải được TLS: dữ liệu truyền đi vẫn là ciphertext (vault đã mã hóa)
- Master password không bao giờ rời khỏi trình duyệt — không thể chặn qua mạng

4.5. Kịch bản: Insider threat (nhân viên TJGuard)

Vì hệ thống zero-knowledge, nhân viên TJGuard không có khả năng kỹ thuật để đọc vault của người dùng, ngay cả với toàn quyền truy cập database và source code.

5. So sánh với các tiêu chuẩn ngành

Tiêu chí	**TJGuard**	**Bitwarden**	**1Password**	**LastPass (cũ)**
KDF	Argon2id [1]	Argon2id	2SKD (PBKDF2)	PBKDF2
AEAD	XChaCha20-Poly1305 [2]	AES-CBC + HMAC	AES-GCM	AES-CBC
Chia sẻ vault	Ed25519 [8] + nacl.box	RSA-2048	P-521 ECDH	AES (no PK)

Zero-knowledge	Có	Có	Có	Không (breach 2022)
Open-source crypto	Có	Có	Không hoàn toàn	Không

TJGuard sử dụng cùng bộ thuật toán với các password manager đầu ngành, với lợi thế XChaCha20-Poly1305 (nonce lớn hơn, an toàn hơn cho môi trường web) và Argon2id với memory cost cao.

6. Thư viện mật mã học sử dụng

Thư viện	**Mục đích**	**Nguồn gốc**
hash-wasm	Argon2id [1] (WebAssembly)	MIT License, widely audited
@stablelib/xchacha20poly1305	XChaCha20-Poly1305 [2]	MIT License
tweetnacl	Ed25519 [8], nacl.box	Public domain, kiểm toán bởi Cure53
Web Crypto API	HKDF-SHA256 [3], CSPRNG [6]	Chuẩn W3C, tích hợp trình duyệt
otplib	TOTP/HOTP [7]	MIT License

Tất cả thư viện mật mã học đều là peer-reviewed open source. TJGuard không triển khai bất kỳ thuật toán mật mã học tự chế nào (no "roll your own crypto").

7. Cơ chế bảo vệ phiên làm việc

7.1. Bộ nhớ trong phiên

Khóa mã hóa (Account Key, Private Key) chỉ tồn tại trong:

- Zustand store (JavaScript heap của tab trình duyệt)
- Không lưu vào localStorage, sessionStorage, hay cookie

7.2. Tự động đăng xuất

Sau 10 phút không có hoạt động (không di chuột, gõ phím, cuộn trang), hệ thống tự động:

- Xóa Account Key và Private Key khỏi Zustand store
- Gọi NextAuth signOut()
- Chuyển người dùng về màn hình đăng nhập

7.3. JWT Session

- Thời hạn JWT: 20 phút
- Sliding window refresh: 5 phút
- Cookie: HTTP-only, Secure, SameSite=Strict

8. Cơ chế khôi phục

8.1. Recovery Key

Recovery Key = Account Key thô (32 bytes) được tải xuống dưới dạng file .txt mã hóa sau khi xác thực OTP. Người dùng phải giữ an toàn file này — TJGuard không thể khôi phục thay cho người dùng nếu mất cả master password lẫn recovery key.

8.2. Đổi Master Password

```
1. Người dùng nhập recovery key + master password mới
2. Client tạo Argon2id(new_password + new_salt) → new_KEK
3. XChaCha20-Poly1305.seal(accountKey, new_KEK) → new_ak_wrapped
4. Gửi (new_kdf_salt, new_ak_wrapped) lên server
5. Vault items không cần mã hóa lại (accountKey không đổi)
```

8.3. Recovery Key Rotation

Khi người dùng chủ động rotate recovery key (Account Key mới):

- Tạo Account Key mới
- Dẫn xuất lại tất cả Item Key (HKDF với accountKey mới)
- Mã hóa lại toàn bộ vault items
- Cập nhật tất cả shared item keys cho người được chia sẻ

9. Kiểm toán và minh bạch

9.1. Các thuật toán đã được kiểm toán độc lập

- Argon2id [1]: Đạt giải nhất PHC 2015, phân tích bởi hàng chục nhóm nghiên cứu độc lập.
- ChaCha20-Poly1305 [2]: RFC 8439, được tích hợp vào TLS 1.3.
- NaCl/TweetNaCl: Kiểm toán bởi Cure53 (công ty bảo mật hàng đầu châu Âu).
- HKDF [3]: NIST SP 800-56C, được dùng trong TLS, Signal Protocol, và hàng ngàn ứng dụng.

9.2. Không "roll your own crypto"

TJGuard không tự triển khai bất kỳ primitive mật mã học nào. Toàn bộ đều dùng các thư viện đã được kiểm toán bởi cộng đồng bảo mật quốc tế.

10. Tiêu chuẩn và tuân thủ

Thuật toán	**Tiêu chuẩn**
Argon2id	IETF RFC 9106 [1]
HKDF-SHA256	NIST SP 800-56C [4], RFC 5869 [3]

XChaCha20-Poly1305	RFC 8439 [2] + XSalsa20 extension
Ed25519 / X25519	RFC 8032 [8], RFC 7748 [5]
TOTP	RFC 6238 [7]
CSPRNG	NIST SP 800-90A [6]

Kết luận

TJGuard xây dựng kiến trúc bảo mật trên nguyên tắc: không tin tưởng bất kỳ ai, kể cả chính chúng tôi. Bằng cách dùng các thuật toán mật mã học hiện đại, đã được kiểm toán, và thực thi zero-knowledge triệt để ở tầng kỹ thuật — không phải chỉ là lời hứa — TJGuard đảm bảo rằng bảo mật của người dùng không phụ thuộc vào việc có tin tưởng nhà cung cấp hay không.

Tài liệu tham khảo

[1] A. Biryukov, D. Dinu, D. Khovratovich, S. Josefsson, “RFC 9106 Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications”, https://www.rfc-editor.org/rfc/rfc9106, September 2021.

[2] Y. Nir, A. Langley, “RFC 8439 — ChaCha20-Poly1305”, https://www.rfc-editor.org/rfc/rfc8439, June 2018.

[3] H. Krawczyk, P. Eronen, “RFC 5869 — HKDF”, https://www.rfc-editor.org/rfc/rfc5869, May 2010.

[4] Elaine Barker, Lily Chen, Richard Davis, “NIST SP 800-56C — KDF”, https://csrc.nist.gov/pubs/sp/800/56/c/r2/final, August 2020.

[5] A. Langley, M. Hamburg, S. Turner, “RFC 7748 — X25519/Curve25519”, https://www.rfc-editor.org/rfc/rfc7748, January 2016.

[6] Elaine Barker, John Kelsey, “NIST SP 800-90A Rev.1 — CSPRNG”, https://csrc.nist.gov/pubs/sp/800/90/a/r1/final, June 2015.

[7] D. M'Raihi, S. Machani, M. Pei, J. Rydell, “RFC 6238 — TOTP”, https://www.rfc-editor.org/rfc/rfc6238, May 2011.

[8] S. Josefsson, I. Liusvaara, “RFC 8032 — Ed25519/EdDSA”, https://www.rfc-editor.org/rfc/rfc8032, January 2017.