

SAKF: A Self-Adaptive Key Framework:

Invisible Key Derivation, a Forward-Secure Ratchet and a Hidden-Bit Key-Confirmation Layer (HTAL-K)

S. Ravichandran Ayyer

Independent Researcher, Puttur, Andhra Pradesh, India

ORCID: [0009-0003-6094-2534](https://orcid.org/0009-0003-6094-2534) · Email: saktheee2009@gmail.com

Abstract

This paper concerns cryptographic keys only, not a new cipher. It presents SAKF, a Self-Adaptive Key Framework in which no key ever crosses the network and every message is protected by a different key. SAKF has four layers. Layer 0 establishes a root secret from a passphrase (PBKDF2-HMAC-SHA256, 600,000 iterations) or an X25519 key agreement. Layer 1, the "invisible key", uses HKDF-SHA256 to derive keys on both sides, bound to the session, a circumstance label, the message counter and the sending time. Layer 2 is a symmetric ratchet that erases each chain key after use, giving forward secrecy, and stores skipped keys so that lost and out-of-order messages are handled. Layer 3, HTAL-K, applies the author's Hidden Text Architecture Layer to a 256-bit key-confirmation tag: in every byte one bit is withheld and never transmitted, and the remaining seven bits are rotated, with position and rotation changing per byte and per message. The framework was implemented in Python and C++ and evaluated with standard test vectors (RFC 4231, RFC 5869), 21 functional and security checks, avalanche tests, the NIST SP 800-22 statistical tests on 51,200 derived keys, and a dedicated evaluation of HTAL-K. All checks passed; the Python and C++ implementations agree on all 22 known-answer values; one bias defect in the HTAL-K rotation was found by the tests and corrected. The paper states plainly what the evaluation does and does not show: HTAL-K provides key confirmation equivalent to a 224-bit truncated MAC but adds no secrecy beyond the key, the ratchet does not recover from a compromise without a Diffie-Hellman step, and no formal proof or independent cryptanalysis has yet been carried out.

Keywords: key derivation; HKDF; key management; forward secrecy; ratchet; key confirmation; Hidden Text Architecture Layer; Kerckhoffs's principle; NIST SP 800-22

1. Introduction

Most failures in practical cryptography are failures of key management rather than of ciphers. A key that travels can be intercepted; a key that never changes exposes every past and future message once it is stolen. This paper asks a narrow question: **can two parties obtain a fresh, verifiable key for every message without ever transmitting a key, and with old keys destroyed after use?**

SAKF answers it by combining three ideas developed by the author with standard, well-analysed primitives:

1. **Invisible keys.** Both parties derive the same key from a shared root and public context; the key is never sent.
2. **Self-adaptive keys.** The key changes with every message and depends on the circumstances (a label) and the sending time.

3. **Hidden bits (HTAL).** Part of the confirmation value is never transmitted; the receiver regenerates it.

The contributions are: (i) a precise four-layer design built only on standard primitives; (ii) reference implementations in Python and C++ that agree byte for byte; (iii) a reproducible evaluation in seven categories; and (iv) an honest analysis that separates what each layer contributes to security from what it does not. The scope is deliberately limited to key generation, key change and key confirmation. Encryption is shown only as a use of the derived keys, with a standard authenticated cipher.

2. Background and Related Work

Key derivation. HMAC (Krawczyk, Bellare and Canetti, 1997; Bellare, Canetti and Krawczyk, 1996) is the standard keyed hash. HKDF (Krawczyk and Eronen, 2010; Krawczyk, 2010) extracts a uniform key from a shared secret and expands it into many keys separated by an "info" context; NIST SP 800-108 (Chen, 2022) gives related recommendations. PBKDF2 (Moriarty, Kaliski and Rusch, 2017) stretches a passphrase to slow guessing.

Keys that are never sent. The one-time passwords used in banking, HOTP and TOTP (M'Raihi et al., 2005; 2011), already derive a value on both sides from a shared secret and a counter or time. SAKF's Layer 1 follows the same principle, using HKDF and binding more context.

Ratchets. The Double Ratchet of the Signal protocol (Perrin and Marlinspike, 2016) combines a symmetric-key ratchet with a Diffie-Hellman ratchet; its security has been analysed formally (Cohn-Gordon et al., 2017; Alwen, Coretti and Dodis, 2019). SAKF's Layer 2 is the symmetric half, including the skipped-key mechanism.

Key agreement and encryption. X25519 (Bernstein, 2006; Langley, Hamburg and Turner, 2016) implements Diffie-Hellman key agreement (Diffie and Hellman, 1976). ChaCha20-Poly1305 (Nir and Langley, 2018) is a standard authenticated cipher. SHA-256 is specified in FIPS 180-4.

Kerckhoffs's principle. A system must remain secure when everything except the key is public (Kerckhoffs, 1883). This principle is used throughout Section 4 to judge each layer, in particular HTAL-K.

Author's earlier work. The Hidden Text Architecture Layer (HTAL) was proposed in the author's earlier work as a bit-splitting method in which part of each character's bits is withheld from transmission and regenerated by the receiver. SAKF applies it, for the first time, to key confirmation.

3. Design of SAKF

Figure 1. The four layers of SAKF

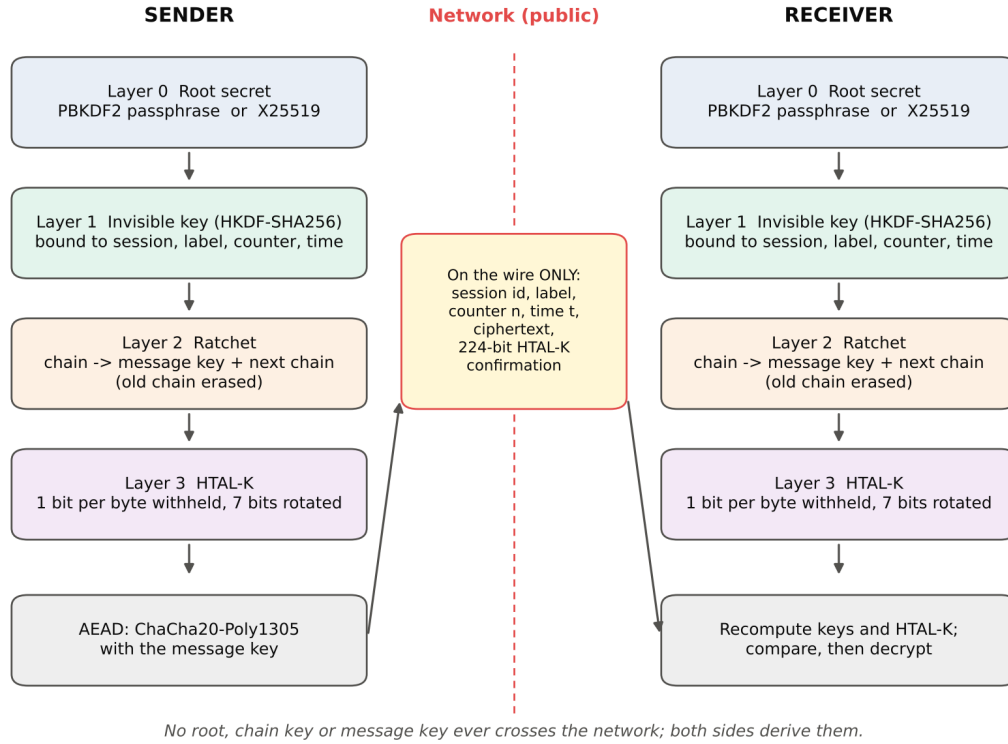


Table 1. What is computed and what is transmitted

Item	Size	Computed by	Transmitted?
Root secret	256 bits	Both parties (Layer 0)	Never
Chain key	256 bits	Both parties (Layer 1, then Layer 2)	Never; erased after each step
Message key, encryption key, MAC key, HTAL key	256 bits each	Both parties (Layers 1 and 2)	Never
Session id, label, counter n, sending time t	Variable; 8 bytes each for n and t	Sender	Yes, in clear; bound to the keys
Ciphertext with Poly1305 tag	Message length + 16 bytes	Sender	Yes
HTAL-K confirmation	224 bits (28 bytes)	Sender; regenerated by receiver	Yes; the 32 withheld bits are never transmitted

3.1 Layer 0: root secret

Two modes are supported. In **Mode P**, both parties hold a passphrase and a salt, and the root is PBKDF2-HMAC-SHA256 with 600,000 iterations. In **Mode E**, each party has an X25519 key pair; the shared Diffie-Hellman value is passed through HKDF-Extract with a hash of both public keys as salt. Mode E removes the need for a pre-shared secret but not the need to authenticate the public keys.

3.2 Layer 1: invisible key derivation

The initial chain key is $\text{chain}_0 = \text{HKDF-Expand}(\text{HKDF-Extract}(\text{"SAKF-v1|root"}, \text{root}), \text{"SAKF-v1|chain|"} + \text{session id} + \text{label}, 32 \text{ bytes})$. For message n sent at time t (nanoseconds), the message key mk is expanded into 96 bytes: $\text{HKDF-Expand}(\text{mk}, \text{"SAKF-v1|msg|"} + n + t + \text{label}) = \text{encryption key} \parallel \text{MAC key} \parallel \text{HTAL key}$. All context fields are length-prefixed, so different contexts cannot produce the same input. The label carries the "circumstances", for example head-office or branch-office; the counter and time make every key unique to its message and moment.

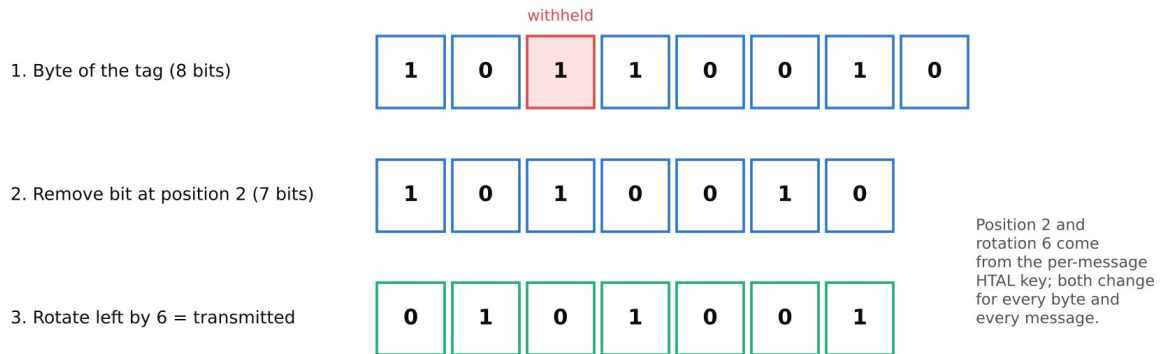
3.3 Layer 2: ratchet

Each step computes $\text{mk} = \text{HMAC}(\text{chain}, 0x01)$ and $\text{chain}' = \text{HMAC}(\text{chain}, 0x02)$, then overwrites the old chain key. The receiver accepts message n only if it is newer than the last message, or if its key is in the store of skipped keys (at most 100). A message older than 300 seconds, a replay, or a message that fails confirmation is rejected, and the receiver's state changes only after a message has been fully verified.

3.4 Layer 3: HTAL-K, hidden bits in the key confirmation

The sender computes a 256-bit confirmation tag, $\text{tag} = \text{HMAC}(\text{MAC key}, \text{"SAKF-v1|confirm|"} + \text{header} + \text{ciphertext})$. From the HTAL key, HKDF derives for each of the 32 bytes a withheld bit position p (0 to 7) and a rotation r (0 to 6). In each byte the bit at position p is removed, the remaining seven bits are rotated left by r , and the $7 \times 32 = 224$ bits are transmitted. The receiver derives the same keys, recomputes the tag and its HTAL-K encoding, and compares in constant time. The withheld bits are never transmitted, and the positions and rotations, the "mutation", change for every byte and every message.

Figure 2. HTAL-K applied to one byte of the confirmation tag



3.5 Use: authenticated encryption

The encryption key is used with ChaCha20-Poly1305; the 12-byte nonce is built from the counter, which is unique because every message key is used once. The header (version, session id, label, counter, time) is authenticated as associated data, so changing the time by one nanosecond invalidates the message.

4. Security Analysis

The analysis assumes that HMAC-SHA256 behaves as a pseudorandom function (PRF), the standard assumption behind HKDF, and that X25519 and ChaCha20-Poly1305 are secure. It is an argument, not a formal proof.

Layer 0. In Mode P, security depends on the passphrase's entropy; PBKDF2 with 600,000 iterations slows each guess (about 0.18 s per guess in Python on the test machine) but cannot rescue a weak passphrase. In Mode E, a passive eavesdropper learns nothing about the root, but an active attacker who can substitute public keys can impersonate either party unless the keys are authenticated.

Layer 1. Under the PRF assumption, keys derived with different context strings are indistinguishable from independent random keys. Because the context is length-prefixed and includes the session, label, counter and time, no two messages share a key. The key never appears on the wire; the evaluation confirms that no 4-byte fragment of any key appears in the transmitted data.

Layer 2. Because each chain key is overwritten, an attacker who steals the current state cannot compute earlier message keys (forward secrecy). The converse does not hold: with a stolen state the attacker can derive all later keys until the root is renewed, because the symmetric ratchet has no Diffie-Hellman step (no post-compromise security). This limitation is demonstrated in the tests.

Layer 3, judged by Kerckhoffs's principle. Assume the attacker knows the full HTAL-K algorithm but not the keys.

1. *Secrecy.* HTAL-K hides nothing that is not already protected by the key. The tag is a pseudorandom value, so its withheld bits are independent of its transmitted bits (measured agreement with a neighbouring bit: 50.14 per cent). The keys themselves are never transmitted in any layer, so HTAL-K does not protect key material against an eavesdropper; it cannot, and does not need to.
2. *Authentication.* The receiver checks 224 bits. HTAL-K is therefore a key-confirmation tag equivalent to a 224-bit truncated MAC whose truncation pattern is keyed; the chance of a random forgery is 2^{-224} .
3. *Mutation.* The rotation is a keyed permutation of seven bits. A permutation of a pseudorandom string is still pseudorandom, so the mutation neither weakens nor strengthens the tag; its value lies in implementing the author's concept faithfully and in making the transmitted bit layout different for every byte and message.
4. *"Rotation in transit."* An earlier description of the concept spoke of bits that keep rotating while travelling. Data in transit is a fixed bit string that no intermediate node changes; an attacker who records it obtains a static copy. In SAKF the mutation is therefore applied once by the sender and reproduced by the receiver, which is the realisable form of the idea.

In short, the security of SAKF rests on the secrecy of the root and on HMAC-SHA256; HTAL-K contributes key confirmation and a concrete implementation of hidden, never-transmitted bits, but not additional secrecy.

5. Methodology

Two independent implementations were written: a Python reference (sakf.py) that includes all layers and the authenticated encryption, and a C++17 implementation (sakf.cpp) of Layers 1 to 3 with its own SHA-256, HMAC and HKDF code. The evaluation (sakf_tests.py) has seven parts:

1. **V: standard test vectors.** HMAC-SHA256 against RFC 4231 test cases 1 and 2; HKDF against RFC 5869 test case 1 (PRK and OKM). The C++ code is also checked against the FIPS 180-4 SHA-256 example.
2. **F: functional and security behaviour.** Both root modes; out-of-order delivery (3, 1, 5, 2) with message 4 lost; replay; key leakage on the wire; forward secrecy and its limit; tampering with ciphertext, confirmation and time; age limit; state after rejection; wrong secret; context separation.
3. **A: avalanche.** For 1,000 random roots each, the Hamming distance between 256-bit encryption keys when one bit of the root changes, one bit of the time changes, the counter advances, or one character of the label changes. The ideal is Binomial(256, 0.5): mean 128, standard deviation 8.
4. **S: statistical quality.** 51,200 consecutive message keys, arranged as 100 sequences of 131,072 bits, tested with eight NIST SP 800-22 tests (Rukhin et al., 2010), using an implementation validated against the worked examples in the standard. Pass criterion: proportion of passing sequences at least 0.9602 and uniformity P-value at least 0.0001. Also byte-value uniformity and the distance between consecutive keys.
5. **H: HTAL-K.** Acceptance of 10,000 genuine confirmations; uniformity of withheld positions and rotations over 320,000 bytes; 100,000 random forgeries; all 224 single-bit changes for 100 messages; independence of withheld bits.
6. **P: performance** of derivation, sending and receiving in Python (median of 2,000 runs).
7. **K: cross-language agreement.** Fixed inputs produce 22 values (chain keys, message keys, encryption, MAC and HTAL keys, tags and confirmations for three messages) that are compared between Python and C++.

All random inputs use a fixed seed, so every number in Section 6 can be reproduced.

6. Results

6.1 Test vectors and cross-language agreement

Table 2. Standard vectors and implementation agreement

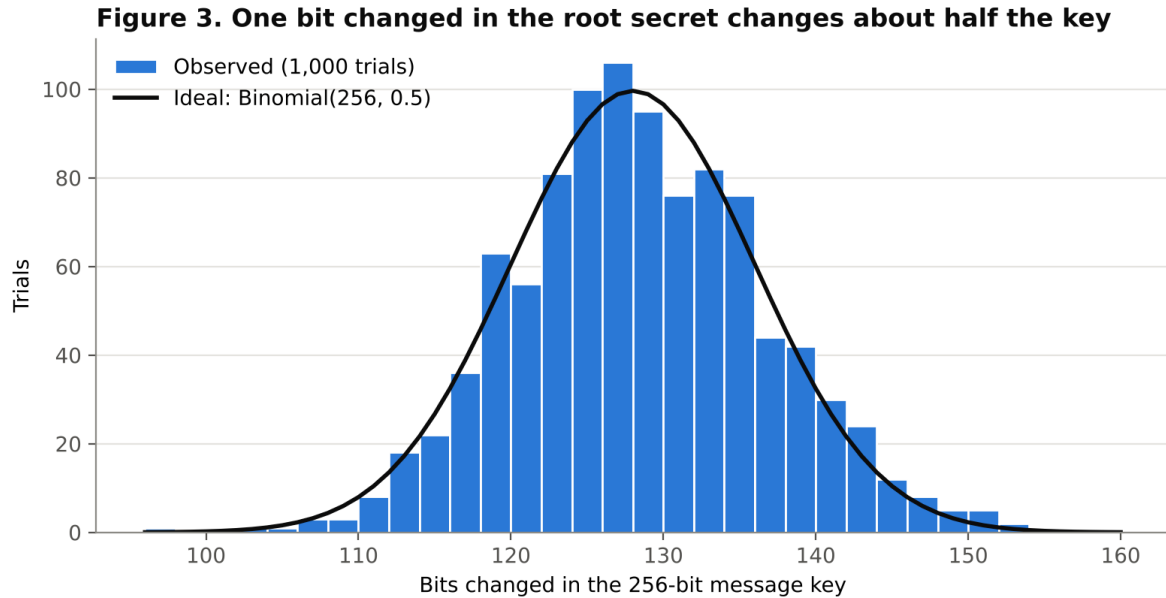
Check	Python	C++
SHA-256 "abc" (FIPS 180-4)	via hashlib	PASS
HMAC-SHA256, RFC 4231 test cases 1 and 2	PASS	PASS
HKDF, RFC 5869 test case 1 (PRK and OKM)	PASS	PASS
22 known-answer values of SAKF (3 messages)	Reference	Identical to Python

6.2 Functional and security behaviour

All 21 checks passed. They show that both root modes give both parties the same root; that out-of-order messages decrypt correctly and a lost message leaves exactly one stored key; that

replays, tampered ciphertexts, tampered confirmations, a time changed by one nanosecond, messages older than 300 seconds and a wrong root are all rejected without changing the receiver's state; that no fragment of any key appears on the wire; that different labels give unrelated chains; and that a stolen current state cannot read old messages. One check deliberately confirms a known weakness: a stolen state can read later messages.

6.3 Avalanche of the key derivation



Mean and spread for all four inputs are in Table 3.

Table 3. Bits changed in the 256-bit encryption key (1,000 trials each)

Change in input	Mean	Standard deviation	Minimum	Maximum
1 bit of the root secret	127.78	7.88	106	150
1 bit of the sending time	128.24	8.08	103	154
Next message (counter $n + 1$)	128.05	7.97	98	156
1 character of the label	127.74	8.06	101	153
Ideal (Binomial 256, 0.5)	128.00	8.00		

Every input change alters about half of the key, with the spread expected of independent random keys.

6.4 Statistical quality of derived keys

Table 4. NIST SP 800-22 tests on 51,200 message keys (100 sequences of 131,072 bits; minimum pass proportion 0.9602)

Test	Sequences passing	Uniformity P-value	Result
Frequency	97/100	0.5955	Pass
Block frequency (M = 128)	100/100	0.0110	Pass
Runs	100/100	0.7598	Pass
Longest run (M = 128)	100/100	0.4373	Pass
Cumulative sums (forward)	97/100	0.7598	Pass
Serial, P-value 1 (m = 5)	100/100	0.2133	Pass
Serial, P-value 2 (m = 5)	99/100	0.6371	Pass
Approximate entropy (m = 5)	100/100	0.7598	Pass

Byte values were uniform (chi-square 239.1 with 255 degrees of freedom, $p = 0.7554$), and consecutive keys differed in 128.03 bits on average (standard deviation 7.99). These tests can reveal gross defects; they cannot prove security.

6.5 HTAL-K evaluation

Figure 4. HTAL-K selections are uniform after the fix

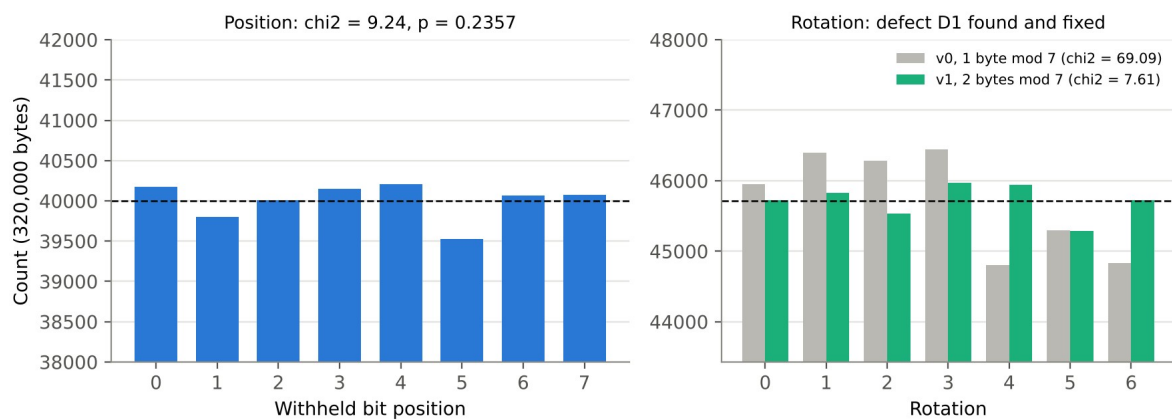


Table 5. HTAL-K layer

Measure	Result
Genuine confirmations accepted	10,000 of 10,000
Withheld-bit positions uniform (320,000 bytes)	chi-square 9.24 (7 df), $p = 0.2357$
Rotations uniform, version 1	chi-square 7.61 (6 df), $p = 0.2684$
Rotations uniform, version 0 (defect D1)	chi-square 69.09 (6 df), p below 0.0001
Random forgeries accepted	0 of 100,000
Single-bit changes rejected	22,400 of 22,400
Withheld bit equal to its neighbour	50.14% (ideal 50%)

Defect D1, found and corrected. In the first version, each rotation was taken as one random byte reduced modulo 7. Because 256 is not a multiple of 7, rotations 0 to 3 occurred slightly more often (37 in 256 against 36 in 256), and the uniformity test detected it (chi-square 69.09). Version 1 reduces a 16-bit value instead, leaving a bias below 0.004 per cent, and the test then passes. The defect did not affect the security argument in Section 4, but it shows the value of testing every design choice, including the small ones.

6.6 Performance

Table 6. Time per operation (Python 3, single core; median of 2,000 runs)

Operation	Time
Derive one message's keys (ratchet step and HKDF)	about 27 microseconds
Send (keys, encryption of 256 bytes, HTAL-K)	about 0.14 ms
Receive (keys, HTAL-K check, decryption)	about 0.15 ms
X25519 key agreement (Mode E)	about 0.12 ms
PBKDF2, 600,000 iterations (Mode P, once per session)	about 0.18 s

The per-message cost is small; the deliberate cost is in PBKDF2, paid once per session. Timings vary by machine and run and are given only as orders of magnitude.

7. Discussion: Strengths, Weaknesses and Open Questions

Table 7. What has been studied and what remains

Aspect	Status in this paper	What remains to be studied
Correctness of primitives	Standard vectors pass in Python and C++	
Agreement between implementations	22 values identical	Third independent implementation
Key never transmitted	Confirmed by design and by test	
Per-message keys bound to time and circumstance	Confirmed (avalanche, context separation)	
Forward secrecy	Confirmed	Formal proof
Recovery after compromise	Not provided	Add a Diffie-Hellman ratchet (as in the Double Ratchet)
Lost and out-of-order messages	Confirmed (store of up to 100 keys)	Resistance to deliberate flooding of the store
Statistical quality of keys	NIST subset passed	Full NIST suite, TestU01
HTAL-K	Uniform, unforgeable at 2^{-224} , no secrecy gain	Benefit in settings where tags are exposed separately
Side channels (timing, memory)	Not evaluated; C++ comparison and erasure are not hardened	Constant-time C++ code; secure memory handling
Clock dependence	300-second window tested	Behaviour under clock drift and network delay
Independent cryptanalysis	None yet	Open review by the community
Formal security proof	None	Reduction to the PRF security of HMAC

Strengths. SAKF keeps keys off the network entirely, gives every message its own key bound to time and circumstance, destroys old keys, tolerates lost and reordered messages, rejects replays and tampering without corrupting its state, and is built from standard primitives whose security is widely studied. Its derivations are fully specified and reproducible across languages.

Weaknesses. Security rests on the root secret: a weak passphrase, or unauthenticated X25519 keys, defeats the framework. Without a Diffie-Hellman ratchet, a stolen state compromises all later messages. Counter and time are sent in clear, revealing metadata. HTAL-K does not add secrecy, and the "rotation in transit" of the original concept is not physically realisable. The implementations have not been hardened against side channels or reviewed by others.

8. Future Work

1. Add a Diffie-Hellman ratchet step for post-compromise security, and post-quantum key agreement (for example ML-KEM) in Layer 0.
2. Write a formal reduction of Layers 1 and 2 to the PRF security of HMAC.
3. Harden the C++ code (constant-time comparison, guaranteed erasure) and add the authenticated encryption.
4. Run the full NIST SP 800-22 suite and other batteries on larger samples.

5. Publish the code for open review and invite cryptanalysis of the framework and of HTAL-K.

9. Conclusion

SAKF shows that the author's concepts of invisible, self-adaptive keys and hidden bits can be built into a coherent key framework from standard, well-understood primitives. In the evaluation, keys never left the device, changed with every message, time and circumstance, behaved statistically like random keys, and survived lost, reordered, replayed and tampered messages; two independent implementations agreed exactly; and the one defect found was corrected. The paper is equally clear about the limits: the framework is only as strong as its root secret, it does not recover from a compromise, HTAL-K confirms keys but adds no secrecy, and formal proof and independent review are still to come. Those limits define the next stage of the work.

References

- [1] Alwen, J., Coretti, S., & Dodis, Y. (2019). The Double Ratchet: Security notions, proofs, and modularization for the Signal protocol. In *Advances in Cryptology – EUROCRYPT 2019*, LNCS 11476, 129–158. Springer.
- [2] Bellare, M., Canetti, R., & Krawczyk, H. (1996). Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO '96*, LNCS 1109, 1–15. Springer.
- [3] Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman speed records. In *Public Key Cryptography – PKC 2006*, LNCS 3958, 207–228. Springer.
- [4] Chen, L. (2022). *Recommendation for Key Derivation Using Pseudorandom Functions* (NIST Special Publication 800-108r1). National Institute of Standards and Technology.
- [5] Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., & Stebila, D. (2017). A formal security analysis of the Signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy*, 451–466.
- [6] Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), 644–654.
- [7] Kerckhoffs, A. (1883). La cryptographie militaire. *Journal des sciences militaires*, 9, 5–38.
- [8] Krawczyk, H. (2010). Cryptographic extraction and key derivation: The HKDF scheme. In *Advances in Cryptology – CRYPTO 2010*, LNCS 6223, 631–648. Springer.
- [9] Krawczyk, H., Bellare, M., & Canetti, R. (1997). *HMAC: Keyed-Hashing for Message Authentication* (RFC 2104). IETF.
- [10] Krawczyk, H., & Eronen, P. (2010). *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)* (RFC 5869). IETF.
- [11] Langley, A., Hamburg, M., & Turner, S. (2016). *Elliptic Curves for Security* (RFC 7748). IETF.
- [12] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., & Ranen, O. (2005). *HOTP: An HMAC-Based One-Time Password Algorithm* (RFC 4226). IETF.
- [13] M'Raihi, D., Machani, S., Pei, M., & Rydell, J. (2011). *TOTP: Time-Based One-Time Password Algorithm* (RFC 6238). IETF.
- [14] Moriarty, K., Kaliski, B., & Rusch, A. (2017). *PKCS #5: Password-Based Cryptography Specification Version 2.1* (RFC 8018). IETF.

- [15] National Institute of Standards and Technology. (2015). *Secure Hash Standard (SHS)* (FIPS PUB 180-4).
- [16] Nir, Y., & Langley, A. (2018). *ChaCha20 and Poly1305 for IETF Protocols* (RFC 8439). IETF.
- [17] Nystrom, M. (2005). *Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512* (RFC 4231). IETF.
- [18] Perrin, T., & Marlinspike, M. (2016). *The Double Ratchet Algorithm*. Signal specification.
- [19] Rukhin, A., Soto, J., Nechvatal, J., et al. (2010). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* (NIST Special Publication 800-22 Rev. 1a).

Declarations

Nature of the study. This paper presents a key-management design and its experimental evaluation. It does not claim a formal security proof.

Code availability. The Python reference implementation (sakf.py), the C++ implementation (sakf.cpp), the evaluation program (sakf_tests.py) and the full results are available from the author and accompany this submission.

Use of AI tools. The author used AI assistants (Claude, by Anthropic) to help formalise the design, write and test the programs and edit the English text. The concepts of invisible keys, self-adaptive keys and HTAL, and the conclusions, are the author's, and the author takes full responsibility for the content.

Competing interests. None.

Funding. None.

About the Author

S. Ravichandran Ayyer is an independent researcher based in Puttur, Andhra Pradesh, India. He holds an M.A. and an M.Phil. in Economics, an M.A. in English, an M.Com., an LL.B. from Sri Venkateswara University, a Postgraduate Diploma in Computer Applications and a Diploma in Cyber Security. He served for about 25 years (1988–2013) in the Tamil Nadu Police, and later taught Economics and English at college level. He has pursued cryptography research independently since 2014 and is a member of the Cryptology Research Society of India. His work covers key management, cipher design and evaluation, as well as economics and legal reform. He has published working papers on SSRN and Zenodo and holds granted and published patents in artificial intelligence and cognitive systems. ORCID: 0009-0003-6094-2534.