Seeded TESLA: Hash-Only Chain Switching

Benjamin Mord

June 2026

1. Summary

In this paper I describe a quantum-resistant extension of TESLA[1], in which an arbitrarily large number of chains is generated from a single random seed. This permits a signer to supply TESLA-based signatures guaranteeing authenticity of future chains long in advance of their use, permitting users (or manufacturers of devices to be later sold to users) to precache validation signatures of future chains long in advance of need. This provides a simple mechanism to avoid use of traditional asymmetric signature algorithms (as in Galileo OSNMA[2]) that is robust to intermittent or even extended communication interruptions. This even permits a receiving device that sits unused for years to promptly reestablish a trust relationship shortly after powering on, or for a device which anticipates signal jamming to arrange for continuous trust relationship maintenance despite signal jamming during chain switch events.

2. Protocol

a. General Definitions

- Let $h(x)$ be a cryptographic hash function with preimage resistance
- Let $h^i(x)$ denote i iterations of h on x
- Let $p(k, x)$ be a pseudorandom function on x keyed by k. To avoid introducing additional unproven assumptions and to simplify response to future cryptanalytic surprises, h can be repurposed. For example if h is SHA-256 then p could be HMAC-SHA-256. If h is SHA-3, then p could be KMAC256.
- Let s be a securely random seed of data, known only to the signer, to serve as the private key

[1] A. Perrig, R. Canetti, J. D. Tygar, D. Song, "The TESLA Broadcast Authentication Protocol," *RSA CryptoBytes*, Vol. 5, No. 2, Summer 2002. https://people.eecs.berkeley.edu/~tygar/papers/TESLA_broadcast_authentication_protocol.pdf

[2] European Union Agency for the Space Programme (EUSPA), *Galileo Open Service Navigation Message Authentication (OSNMA) Signal-In-Space Interface Control Document*, Issue 1.1, October 2023. https://www.gsc-europa.eu/sites/default/files/sites/all/files/Galileo_OSNMA_SIS_ICD_v1.1.pdf

- Let $a(k, m)$ be a message authentication code generated with key k on message m. For example, $a(k, m)$ might use the same algorithm as $p(k, x)$.
- Let $K(i) = h^{l-(i \bmod l)}(p(s, \lfloor i/l \rfloor))$ for $i \in \mathbb{N}_0$ be the ith key that signer will use for signing data, where the design constant l is called "chain length"
- Let $K(0)$ be referred to as the signer's "public key".
- Let $S(i) : \mathbb{N} \to \mathbb{R}$ be a monotonically increasing "key publication schedule" to which the signer publicly commits. In particular, the signer commits to never publish key i earlier than $S(i)$, where $S(i)$ is in seconds since the unix epoch. (Publication delays, however, are acceptable.)
- Let $S^{-1}(t) : [S(1), \infty) \to \mathbb{N}$ be defined s.t. $S^{-1}(t) = i$ for $i \in \mathbb{N}$, $S(i) \leq t < S(i+1)$
- Let the signer commit to only signing with key $K(i)$ where $i \geq S^{-1}(t) + b$ for a publicly chosen design constant $b \in \mathbb{N}$

b. Signer Operation

1. Seed Generation: The signer generates (and keeps secret) the random seed s
2. Public Key Disclosure: The signer discloses $K(0)$ to verifiers, who are assumed to know it authentically
3. Schedule Disclosure: The signer discloses schedule S to verifiers, who are assumed to know it authentically
4. Key Disclosures: On or after $S(i)$ for some $i \in \mathbb{N}$, the signer discloses the ordered pair $(i, K(i))$ to verifiers, so that they may then validate any signatures made with $K(i)$
5. Chain Signatures: at a time earlier than $S(i-b)$ for $i \in \mathbb{N}$ and $j \in \mathbb{N}$, the signer discloses the 4-tuple $(i, c, h(K(c)), a(K(i), h(K(c))))$ where $c = jl$ (recall l is the chain length)
6. Message Signatures: Whenever the signer wishes to sign message m, the signer publicly discloses the ordered triple $(i, m, a(K(i), m))$ with an integer $i \geq S^{-1}(t) + b$. (To enable earliest possible verification, signer chooses $i = S^{-1}(t) + b$.)

c. Verifier Operation

Verifier maintains four lists:

1. VK: Verified keys
2. VCS: Verified chain signatures
3. UCS: Unverified chain signatures
4. UMS: Unverified message signatures

Verifiers also maintain knowledge of schedule $S(i)$ (and thus also $S^{-1}(t)$)

A verifier considers that a key $K(a)$ can be derived from a key $K(b)$ iff $\lfloor a/l \rfloor = \lfloor b/l \rfloor$ and $a \leq b$, in which case $K(a) = h^{b-a}(K(b))$

Verifier proceeds as follows:

1. Public Key Disclosure: verifier initializes VK to $\{(0, K(0))\}$
2. Key Disclosure (mid chain): Upon receiving a key disclosure $(i, K(i))$ such that $i \bmod l \neq 0$, verifier checks VK for $K(j)$ where $\lfloor i/l \rfloor * l \leq j < i$. If found, and if $K(j) = h^{i-j}(K(i))$, then verifier replaces $K(j)$ with $K(i)$ in VK. Otherwise, verifier discards $K(i)$.
3. Key Disclosure (chain start): Upon receiving a key disclosure such that $i \bmod l = 0$ and $i \neq 0$, verifier checks VCS for a chain signature $(k, c, h(K(c)), a(K(k), h(K(c))))$ such that $c = i$. If found, verifier confirms if $h(K(i)) = h(K(c))$. If so, verifier adds $K(i)$ to VK, else verifier discards $K(i)$.
4. Chain Signature Receipt: Upon receiving a chain signature $(i, c, h(K(c)), a(K(i), h(K(c))))$, if $S(i) > t + d$ where t is the current time and d is a verifier-specific design constant to accommodate clock drift, then verifier adds the chain signature to UCS.
5. Chain Signature Verification: If there is a chain signature C_s in UCS for which $S(i) < t$ where t is the current time, and if there is a key in VK from which $K(i)$ can be derived, verifier checks if $a(K(i), h(K(c)))$ is as asserted in C_s. If assertion is correct, verifier moves C_s to VCS. If assertion is incorrect, verifier discards C_s.
6. Message Signature: Upon receiving a message signature $(i, m, a(K(i), m))$, verifier confirms $S(i) > t + d$ where t is the current time and d is the clock-drift accommodation constant. If so confirmed, verifier adds the message signature to UMS, else verifier discards it.
7. Message Signature verification: If a message signature exists in UMS for which a key in VK exists from which $K(i)$ can be derived, then verifier confirms if $a(K(i), m)$ is as asserted by the message signature. If so, verifier processes m as valid. Verifier removes the message signature from UMS.