

ATESO-1 SOVEREIGN SILICON HARDWARE LEDGER AND CONTACTLESS AMBIENT SENTRY

Title of Invention

ATESO-1 Sovereign Silicon Hardware Ledger and Contactless Ambient Sentry with Monotonic Temporal Hash Chain, SRAM-PUF Zero-Extraction Keying, and Software-Only Reverse Sinkhole Detection

Cross-Reference to Related Applications

This application is a continuation-in-part of, and claims the benefit of priority under 35 U.S.C. § 120 and 35 U.S.C. § 119(e) to, U.S. Provisional Application Nos. PROV-001 through PROV-013, each filed by the same inventive entity and each entitled in part “ManyMoats Sovereign Ledger Infrastructure” or “ATESO Secure Ledger Architecture,” the entire contents of each of which are hereby incorporated by reference in their entirety for all purposes.

For the avoidance of doubt, the present specification is structured to be filed as a provisional application under 37 CFR § 1.53(g) with a zero-dollar basic filing fee where the applicant qualifies as a micro entity or small entity, and to provide a twelve-month priority umbrella under 35 U.S.C. § 119(e) for one or more later-filed non-provisional applications. Under 37 CFR § 1.53(g), the present provisional application may be filed without formal patent claims and without an executed inventor declaration, while preserving the priority date of all subject matter disclosed herein for a subsequent non-provisional application filed within twelve months. The formal claims set forth below are provided for the purpose of establishing the full intended scope of the provisional disclosure and to support enablement and written description under 35 U.S.C. § 112(a) in the later-filed non-provisional application.

Technical Field

The present disclosure relates generally to tamper-evident cryptographic ledger hardware, physically unclonable function (PUF) based device identity, and wireless network security. More particularly, the present disclosure relates to an integrated circuit coprocessor comprising an irreversible monotonic temporal counter, a pipelined SHA-256 hash chain register, an SRAM PUF zero-extraction key engine, and a software-only ambient radio-frequency (RF) reverse sinkhole detection system that uses microsecond round-trip time (RTT) jitter and Doppler spread to distinguish physically co-present gateways from remotely spoofed or sinkholed network nodes.

Background of the Invention and Prior Art Limitations

Software-only ledgers and cryptographic timestamping systems rely on host operating system timekeeping, software key storage, and software execution integrity. In such systems, a system clock may be set backwards by NTP spoofing, clock skew, manual alteration, or by a hypervisor-interposed write to a time register. A private key stored in RAM, flash, or a host file system may be extracted through memory side channels, root privileges,

JTAG probing, or physical bus sniffing. An adversary with hypervisor or root access may pause, rewind, or single-step execution between ledger state updates, thereby undermining the integrity of a cryptographic receipt chain.

Prior hardware secure elements, such as discrete Trusted Platform Modules (TPMs) and secure microcontrollers, provide some protection. However, they typically store private keys in non-volatile memory that may be extracted by focused ion beam probing or side-channel analysis. They also rely on external time sources or on internal real-time clocks that may be reset by logical commands. Furthermore, prior contactless or wireless authentication systems often rely on received signal strength indicator (RSSI) alone, which is easily spoofed by a high-gain transmitter or an active relay. There remains a need for a silicon-enforced ledger that makes temporal regression physically impossible, that derives device keys from manufacturing-level physical randomness without storing them in persistent non-volatile memory, and that detects network sinkhole or relay attacks through microsecond-scale round-trip time jitter and ambient Doppler spread.

Summary of the Invention

The present disclosure provides an ATESO-1 integrated circuit coprocessor. In one aspect, an ATESO-1 device includes a hardware monotonic counter, a chained hash register, and a pipelined SHA-256 engine. The monotonic counter transitions according to:

$$C_{n+1} = C_n + \Delta C, \quad \Delta C \geq 1,$$

where $C_n, C_{n+1} \in \mathbb{N}_{64}$ are 64-bit counter values. A write attempt that would load a value X satisfying $X \leq C_n$ is rejected, and an irreversible temporal-regression fault latch is set. The hash chain transitions according to:

$$H_k = \text{SHA} - 256(H_{k-1} \parallel C_k \parallel \text{Evt}_k \parallel \kappa_k),$$

where H_{k-1} is the prior receipt hash, C_k is the current monotonic counter value, Evt_k is an event payload, and κ_k is a sequential transaction nonce. In a preferred embodiment, the pipelined SHA-256 engine completes the computation and latches the new receipt hash into a hardware hash-state register in less than 10 nanoseconds.

In another aspect, the present disclosure provides a zero-extraction device identity and signing engine. An SRAM physical unclonable function captures the uninitialized power-up state of a 1024-cell cross-coupled SRAM array. A fuzzy extractor circuit computes:

$$sk_{\text{device}} = \text{Rep}(W_{\text{SRAM}}, H_{\text{helper}}),$$

where W_{SRAM} is the power-up pattern, H_{helper} is helper data, and sk_{device} is a device private key. The private key is reconstituted on demand, used for Ed25519 public key derivation and signing, and is never stored in persistent non-volatile memory. In a preferred embodiment, the private key is reconstituted in less than 20 microseconds.

In another aspect, the present disclosure provides a software-only reverse sinkhole ambient RF sentry. A standard wireless transceiver samples ambient RF frames. For each frame or transaction, a round-trip time is measured:

$$\Delta t_{\text{RTT},i} = t_{\text{ACK},i} - t_{\text{TX},i} - \Delta t_{\text{proc},i},$$

where $t_{TX,i}$ is the transmission time, $t_{ACK,i}$ is the acknowledgement time, and $\Delta t_{proc,i}$ is the reported processing delay. A jitter dispersion metric is computed as:

$$\sigma_{\Delta t_{RTT}} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (\Delta t_{RTT,i} - \bar{\Delta t_{RTT}})^2}.$$

A multipath variance metric is computed as:

$$\sigma_{\tau}^2 = \frac{1}{N} \sum_{i=1}^N (\tau_i - \hat{\tau}_i)^2,$$

where τ_i is the measured propagation delay and $\hat{\tau}_i$ is the expected propagation delay from a calibrated baseline. A remote sinkhole or relay attack is detected when a candidate gateway's measured jitter dispersion, multipath variance, or Doppler consistency falls outside a calibrated legitimacy region.

The present specification also provides formal patent claims in three groups: Claims 1-10 directed to the hardware monotonic clock and chained hash register coprocessor; Claims 11-20 directed to the PUF device identity and zero-extraction key protocol; and Claims 21-30 directed to the software-only reverse sinkhole ambient RF sensing and gateway jitter sentry.

Detailed Description of Preferred Embodiments

Definitions and Notation

For purposes of the present disclosure, the following terms have the meanings set forth below. “Monotonic” means non-decreasing in numerical value over time; a “monotonic counter” is a counter that cannot be written backwards, decremented, or overwritten with a value less than or equal to its current value. “Temporal regression” means any operation that would cause a counter or timestamp to move backwards in logical time. “Cryptographic hash” means a deterministic one-way function such as SHA-256, SHA-384, SHA-512, SHA-3, BLAKE2, or BLAKE3, with SHA-256 being preferred. “Nonce” means a sequential transaction nonce κ_k that increases for each ledger event. “PUF” means physical unclonable function. “Fuzzy extractor” means an error correction and randomness extraction circuit that converts a noisy PUF pattern into a stable cryptographic key using helper data. “Zero-extraction” means that a private key is generated, used, and zeroized entirely within a hardware security boundary without ever being placed on an external bus or stored in non-volatile memory. “Reverse sinkhole” means a network attack in which a remote adversary impersonates a local gateway or lures client devices to authenticate to a false network; the ATESO-1 reverse sinkhole sentry detects such attacks by requiring microsecond-scale round-trip time jitter and Doppler consistency sufficient to prove physical co-presence.

Embodiment 1: Monotonic Temporal Counter and Chained Hash Register Coprocessor

In a first preferred embodiment, an ATESO-1 integrated circuit includes a RISC-V RV32IMC processor core, an open core such as CV32E, a host interface block, a hardware ledger accelerator, an SRAM PUF key generator, and a tamper-evident eFuse security latch. The ledger accelerator includes a pipelined SHA-256 engine, a 64-bit monotonic counter, and a hash-state register. The host interface is a high-speed SPI slave interface operating at up to 50 MHz.

The monotonic counter is implemented as a 64-bit hardware register. The counter is advanced by an internal oscillator, a hardware event signal, or a command from the host processor. The counter transition is:

$$C_{n+1} = C_n + \Delta C, \quad \Delta C \geq 1.$$

A hardware comparator rejects any write cycle that attempts to load a value X where $X \leq C_n$. In a preferred embodiment, the rejection is performed by a 64-bit magnitude comparator coupled to the counter register. When the comparator detects a temporal regression attempt, the circuit sets an irreversible fault latch named `FAULT_TEMPORAL_REGRESSION`. In one embodiment, the fault latch is an eFuse or antifuse cell that cannot be reset once blown. In another embodiment, the fault latch is a persistent security register that requires a physical destructive reset to clear. The counter cannot be decremented, and the only allowed write direction is forward.

In a preferred embodiment, the monotonic counter and hash-state register are updated in a single hardware transaction. The event payload Evt_k is loaded into an event input buffer. The previous hash H_{k-1} is held in a hardware hash-state register. The current counter value C_k is read from the monotonic counter. The transaction nonce κ_k is read from a nonce register that increments for every sealed event. The hash engine computes:

$$H_k = \text{SHA-256}(H_{k-1} \parallel C_k \parallel Evt_k \parallel \kappa_k).$$

The new hash H_k is latched into the hash-state register, replacing H_{k-1} . Because the previous hash H_{k-1} is destroyed by the latch operation, and because the monotonic counter cannot be decremented, state rollback is mathematically and physically impossible.

In a preferred embodiment, the SHA-256 engine is a dedicated hardware pipeline with at least 64 round-processing stages. Each stage includes a hardware register for the working state and hardware circuits for message schedule expansion, Σ functions, Majority, Choice, and modular addition. The 64-round pipeline is fully unrolled or deeply pipelined, with a critical path sufficiently short that the hash computation and latch operation complete in less than 10 nanoseconds. In an alternative embodiment, the hash engine includes four pipeline stages per SHA-256 round, with a clock period of less than 1 nanosecond and a total initiation interval of less than 10 nanoseconds. In yet another embodiment, the hash engine includes a dual-issue systolic array that processes two independent hash chains simultaneously.

Exemplary Verilog RTL for the monotonic counter is provided below.

```
// ateso_monotonic_clock.v
// Irreversible monotonic timestamp counter with temporal regression fault latch.
module ateso_monotonic_clock (
    input wire      clk,
    input wire      rst_n,
    input wire      write_en,
    input wire [63:0] write_data,
    input wire      advance_en,
    input wire [63:0] advance_delta,
    output reg [63:0] monotonic_counter,
    output reg      fault_temporal_regression
);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            monotonic_counter <= 64'h0000000000000000;
            fault_temporal_regression <= 1'b0;
        end
    end
endmodule
```

```

        end else begin
            if (write_en && advance_en) begin
                // Illegal simultaneous write and advance; treat as temporal regression.
                fault_temporal_regression <= 1'b1;
            end else if (write_en) begin
                if (write_data <= monotonic_counter) begin
                    // Reject backward or same-value write.
                    fault_temporal_regression <= 1'b1;
                end else begin
                    monotonic_counter <= write_data;
                end
            end else if (advance_en) begin
                monotonic_counter <= monotonic_counter + advance_delta;
            end
        end
    end
end
endmodule

```

Exemplary Verilog RTL for the chained hash pipeline is provided below.

```

// sha256_chain_pipeline.v
// Pipelined SHA-256 ledger receipt hash engine.
module sha256_chain_pipeline (
    input wire      clk,
    input wire      rst_n,
    input wire      start,
    input wire [255:0] previous_hash,
    input wire [63:0] monotonic_counter,
    input wire [255:0] event_payload,
    input wire [63:0] transaction_nonce,
    output reg [255:0] new_hash,
    output reg      hash_done,
    output reg      hash_fault
);
    wire [255:0] message_word;
    assign message_word = {previous_hash[255:0], monotonic_counter[63:0],
        event_payload[255:0], transaction_nonce[63:32]};

    // Sixty-four-stage unrolled SHA-256 compression pipeline.
    // Each stage includes round registers, K constants, W expansion, Maj, Ch, Sigma0,
    // Sigma1.
    wire [255:0] stage_out [0:64];
    wire [31:0] w [0:63]; // expanded message schedule
    assign stage_out[0] = previous_hash;

    genvar i;
    generate
        for (i = 0; i < 64; i = i + 1) begin : sha_round_stages
            sha256_round_stage #(.ROUND(i)) round_i (
                .clk(clk),
                .rst_n(rst_n),
                .state_in(stage_out[i]),
            )
        end
    endgenerate
endmodule

```

```

        .w_in(w[i]),
        .k_const(K_CONST[i]),
        .state_out(stage_out[i+1])
    );
end
endgenerate

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        new_hash <= 256'h00000000000000000000000000000000;
        hash_done <= 1'b0;
        hash_fault <= 1'b0;
    end else if (start) begin
        new_hash <= stage_out[64];
        hash_done <= 1'b1;
        hash_fault <= 1'b0;
    end else begin
        hash_done <= 1'b0;
    end
end
endmodule

```

Exemplary SPI slave command interface logic is provided below.

```

// spi_slave.v
// SPI slave command interface with register map.
module spi_slave (
    input wire        sclk,
    input wire        cs_n,
    input wire        mosi,
    output reg        miso,
    input wire [7:0]  command,
    output reg [7:0]  status,
    output reg [31:0] read_data
);
    // Commands:
    // 0x01 CMD_READ_CLOCK
    // 0x02 CMD_SEAL_EVENT
    // 0x03 CMD_GET_PUBKEY
    // 0x04 CMD_SIGN_STATE
    // 0x0F CMD_GET_HEALTH
endmodule

```

A preferred register map for the ATESO-1 ledger accelerator is set forth in Table 1.

Address	Register Name	Width	Access	Description
0x00	MONOTONIC_COUNTER_LOW	32-bit	R	Low word of 64-bit monotonic counter
0x04	MONOTONIC_COUNTER_HIGH	32-bit	R	High word of 64-bit monotonic counter
0x08	HASH_STATE	256-bit	R	Current receipt hash H_k
0x28	EVENT_PAYLOAD	256-bit	W	Event payload Evt_k
0x48	TRANSACTION_NONCE	64-bit	R/W	Sequential nonce κ_k
0x50	COMMAND_REGISTER	8-bit	W	Host command byte
0x54	FAULT_STATUS	32-bit	R	Tamper latches, oscillator health, voltage margin
0x58	PUBLIC_KEY	256-bit	R	Ed25519 public key derived from SRAM PUF

In a preferred embodiment, the command byte 0x02 (CMD_SEAL_EVENT) causes the ledger accelerator to perform the following hardware operations atomically: read current counter value C_k ; read nonce κ_k ; latch event payload Evt_k ; perform SHA-256 hash; latch new hash H_k ; increment nonce; return new hash. The command byte 0x01 (CMD_READ_CLOCK) reads the current monotonic counter value without advancing the hash chain. The command byte 0x03 (CMD_GET_PUBKEY) returns the public key derived from the SRAM PUF. The command byte 0x04 (CMD_SIGN_STATE) signs the current hash-state value with the PUF-derived private key and returns a 64-byte Ed25519 signature. The command byte 0x0F (CMD_GET_HEALTH) returns a status word containing oscillator health, tamper fault latch states, voltage margin, and PUF availability.

Embodiment 2: SRAM PUF Device Identity and Zero-Extraction Key Protocol

In a second preferred embodiment, an ATESO-1 integrated circuit includes a 1024-cell cross-coupled SRAM PUF array. The PUF array is co-located with the ledger accelerator in a single security island. At power-up, the cross-coupled SRAM cells settle into a random pattern $\mathbf{W}_{SRAM}$ determined by microscopic threshold-voltage variations, parasitic capacitance mismatches, and gate oxide variability. A fuzzy extractor circuit receives $\mathbf{W}_{SRAM}$ and stored helper data $\mathbf{H}_{helper}$, and computes:

$$sk_{device} = \text{Rep}(\mathbf{W}_{SRAM}, \mathbf{H}_{helper}).$$

The fuzzy extractor includes an error correction stage and a randomness extraction stage. In one embodiment, the error correction stage is a BCH decoder, Reed-Muller decoder, or convolutional decoder, and the randomness extraction stage is a cryptographic hash or a universal hash. The helper data $\mathbf{H}_{helper}$ may be stored on-chip in a protected non-volatile memory, or off-chip in a publicly readable manner. The helper data alone does not reveal sk_{device} .

The private key sk_{device} is used by an Ed25519 engine to compute a public key pk_{device} . The public key is made available through the `CMD_GET_PUBKEY` command. The private key is used by the `CMD_SIGN_STATE` command to sign the current ledger state. The private key never exists in flash, RAM outside the PUF security island, JTAG-accessible memory, or external memory. After signing, the private key is zeroized by hardware zeroization logic. In a preferred embodiment, the private key is reconstituted in less than 20 microseconds from power-up or from a key-generation command.

If the ATESO-1 device is decapped, probed with a focused ion beam, or subjected to voltage or temperature tampering, the parasitic capacitance of the SRAM PUF cells changes. This change alters the threshold voltages and the power-up pattern, permanently destroying the derived private key. In one embodiment, a decapsulation sensor, laser sensor, or active shield triggers a zeroization command and blows an eFuse tamper latch.

Exemplary Verilog RTL for the SRAM PUF key generation controller is provided below.

```
// sram_puf_keygen.v
// SRAM PUF key generation and zeroization controller.
module sram_puf_keygen (
    input wire      clk,
    input wire      rst_n,
    input wire      start,
    input wire [1023:0] sram_powerup_pattern,
    input wire [255:0] helper_data,
    output reg [255:0] ed25519_public_key,
    output reg      key_ready,
    output reg      key_zeroized,
    output reg      puf_fault
);
    wire [255:0] device_private_key;
    wire [255:0] extracted_key;
    fuzzy_extractor #(.PUF_WIDTH(1024), .KEY_WIDTH(256)) fe (
        .clk(clk),
        .rst_n(rst_n),
        .pattern(sram_powerup_pattern),
        .helper(helper_data),
        .key(extracted_key)
    );

    ed25519_pubkey_gen pubgen (
        .clk(clk),
        .rst_n(rst_n),
        .private_key(device_private_key),
        .public_key(ed25519_public_key)
    );

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            key_ready <= 1'b0;
            key_zeroized <= 1'b1;
            puf_fault <= 1'b0;
        end else if (start) begin
            // Key is generated, used, and then zeroized.

```

```

        key_ready <= 1'b1;
        key_zeroized <= 1'b0;
        puf_fault <= 1'b0;
    end else if (key_ready) begin
        key_zeroized <= 1'b1;
        key_ready <= 1'b0;
    end
end
endmodule

```

Embodiment 3: Software-Only Reverse Sinkhole Ambient RF Sensing and Gateway Jitter Sentry

In a third preferred embodiment, the present disclosure provides a software-only ambient RF sensing method that detects remote sinkhole or relay attacks against a wireless gateway. The method may be implemented as software instructions stored in a non-transitory computer-readable medium and executed by a standard wireless gateway processor, such as a RISC-V processor, an ARM processor, or an x86 processor coupled to a standard WiFi, Bluetooth Low Energy, IEEE 802.15.4, or ultra-wideband transceiver. No dedicated RF hardware beyond the standard transceiver is required.

The reverse sinkhole sentry operates as follows. During a baseline calibration window, a first station receives a plurality of ambient RF frames from a second station that is known to be physically co-present. For each frame or transaction i , the first station measures a round-trip time:

$$\Delta t_{RTT,i} = t_{ACK,i} - t_{TX,i} - \Delta t_{proc,i},$$

where $t_{TX,i}$ is the time of transmission of a poll frame, $t_{ACK,i}$ is the time of receipt of an acknowledgement frame, and $\Delta t_{proc,i}$ is the processing delay reported by the second station. The first station computes a jitter dispersion metric:

$$\sigma_{\Delta t_{RTT}} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (\Delta t_{RTT,i} - \bar{\Delta t_{RTT}})^2}.$$

The first station also computes a carrier frequency offset, or Doppler shift, for each frame:

$$\Delta f_{Doppler,i} = \frac{f_c}{c} \frac{d_i - d_{i-1}}{\Delta t},$$

where f_c is the carrier frequency, c is the speed of light, Δt is the sample interval, and d_i is the estimated propagation distance at sample i . A multipath variance metric is computed as:

$$\sigma_{\tau}^2 = \frac{1}{N} \sum_{i=1}^N (\tau_i - \hat{\tau}_i)^2,$$

where τ_i is the measured propagation delay and $\hat{\tau}_i$ is the expected propagation delay from a calibrated baseline. In a preferred embodiment, the baseline calibration window is selected only during a low-risk time period, such as during initial physical setup, when a trusted human operator confirms physical co-presence, or when a hardware ledger receipt signed by the ATESO-1 device is present.

During subsequent operation, when a candidate gateway asserts an identity or attempts to route ledger traffic, the first station computes candidate metrics for the candidate gateway. The first station flags the candidate

gateway as a reverse sinkhole if the candidate metrics fall outside a calibrated legitimacy region. In a preferred embodiment, the legitimacy region is a multi-dimensional feature space defined by:

$$S = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \mu)^2}$$

What is claimed is:

1. An apparatus for generating tamper-evident monotonic state in a silicon hardware ledger, comprising: a monotonic counter comprising a non-volatile storage array and an increment circuit, the increment circuit configured to advance a counter value represented in the non-volatile storage array in only one direction in response to an increment command; a chained hash register comprising a hash state register and a hash engine, the hash engine coupled to the monotonic counter and the hash state register, wherein the hash engine is configured to compute a current hash state by applying a cryptographic hash function to a concatenation that includes at least a previous hash state stored in the hash state register and the counter value, and to write the current hash state into the hash state register; and an interface coupled to the monotonic counter and the chained hash register and configured to expose the counter value and the current hash state to a requesting circuit while preventing direct external write access to the hash state register except through the hash engine.
2. The apparatus of claim 1, wherein the non-volatile storage array comprises a plurality of one-time programmable memory cells arranged in monotonic banks, and the increment circuit is configured to permanently program at least one not-yet-programmed cell for each increment command.
3. The apparatus of claim 1, further comprising an atomic command sequencer configured to execute an increment-and-hash operation as an atomic sequence that advances the counter value and updates the hash state register without allowing an intervening external access.
4. The apparatus of claim 1, further comprising an initialization circuit configured to initialize the hash state register with a device-unique initialization value obtained from a physically unclonable function without exposing the device-unique initialization value through the interface.
5. The apparatus of claim 1, further comprising a verification circuit configured to receive an externally supplied expected hash chain value, recompute an expected current hash state by iterating the cryptographic hash function over a supplied counter sequence from a referenced hash state, and assert a tamper signal when the recomputed expected current hash state does not match the current hash state.
6. The apparatus of claim 1, wherein the hash engine is configured to implement SHA-256, SHA-3, or a keyed hash message authentication code variant.
7. The apparatus of claim 1, further comprising a power-glitch detector coupled to the monotonic counter and the hash state register, the power-glitch detector configured to block the increment circuit and the hash engine when a supply voltage falls outside a defined operating window.
8. A method for generating tamper-evident monotonic state in a silicon hardware ledger, comprising: receiving an increment command at a monotonic counter; advancing a counter value stored in a non-volatile storage array in only one direction in response to the increment command; reading a previous hash state from a hash state register; computing a current hash state by applying a cryptographic hash

function to at least the previous hash state and the advanced counter value; storing the current hash state in the hash state register; and outputting the advanced counter value and the current hash state, wherein direct external write access to the hash state register is prevented.

9. The method of claim 8, further comprising performing the receiving, advancing, reading, computing, storing, and outputting as an atomic sequence that cannot be interrupted by an external command.
10. The method of claim 8, further comprising initializing the hash state register with a device-unique bit string derived from a physically unclonable function before receiving the increment command, wherein the device-unique bit string is not output.
11. A method for zero-extraction device identity generation and signing in an integrated circuit, comprising: powering up a static random-access memory physical unclonable function (SRAM PUF) array so that the SRAM PUF array adopts a power-up pattern; reading the power-up pattern as a raw SRAM PUF response; within a security boundary that isolates a processing core and a volatile register file from external access, applying an error-correction decoder to the raw SRAM PUF response using helper data stored in non-volatile memory to produce a stable device-unique bit string; deriving a private signing key from the stable device-unique bit string within the security boundary; computing a public key corresponding to the private signing key within the security boundary; exporting the public key; and signing a challenge message with the private signing key within the security boundary to produce a signature, wherein the raw SRAM PUF response, the stable device-unique bit string, and the private signing key are not exported outside the security boundary.
12. The method of claim 11, further comprising, during an enrollment phase: reading the SRAM PUF array multiple times to obtain multiple raw SRAM PUF responses; selecting stable bits from the multiple raw SRAM PUF responses; and storing the helper data derived from the multiple raw SRAM PUF responses in the non-volatile memory, wherein the helper data does not reveal the stable device-unique bit string.
13. The method of claim 11, wherein the error-correction decoder comprises a BCH decoder, a Reed-Muller decoder, or a low-density parity-check decoder.
14. The method of claim 11, wherein deriving the private signing key comprises applying a key derivation function to the stable device-unique bit string and a purpose label, and further comprising applying a second key derivation function to produce one or more device authentication keys independent from the private signing key.
15. The method of claim 11, further comprising, after signing the challenge message, erasing the raw SRAM PUF response, the stable device-unique bit string, and the private signing key from the volatile register file within the security boundary.
16. The method of claim 11, wherein signing the challenge message uses an Edwards-curve digital signature algorithm or an elliptic curve digital signature algorithm executed by a hardware accelerator within the security boundary.
17. The method of claim 11, further comprising: measuring an environmental condition of the integrated circuit; comparing the measured environmental condition to a range used during enrollment; and blocking

the deriving or signing when the measured environmental condition falls outside the range.

18. An integrated circuit for zero-extraction device identity and signing, comprising: an SRAM PUF array; non-volatile memory storing helper data; a security boundary enclosing a processing core, a hardware signature engine, and a volatile register file; a fuzzy extractor circuit configured to receive a raw SRAM PUF response from the SRAM PUF array after power-up, apply an error-correction decoder to the raw SRAM PUF response using the helper data, and produce a stable device-unique bit string within the security boundary; a key derivation circuit configured to derive a private signing key from the stable device-unique bit string within the security boundary; and an output interface configured to export a public key corresponding to the private signing key and a signature generated over a challenge message using the private signing key, wherein the integrated circuit is configured to prevent the raw SRAM PUF response, the stable device-unique bit string, and the private signing key from being exported outside the security boundary.
19. The integrated circuit of claim 18, wherein the helper data is stored in one-time programmable memory and is generated during enrollment from multiple reads of the SRAM PUF array, the helper data being configured not to reveal the stable device-unique bit string.
20. The integrated circuit of claim 18, further comprising the apparatus of claim 1, wherein the hardware signature engine is configured to bind a current hash state from the chained hash register into the signature.
21. A method for software-only ambient radio-frequency sensing and reverse sinkhole detection, comprising: receiving, by a software-defined radio receiver of a wireless station, a plurality of ambient RF signals over a monitored band without transmitting; extracting, by a processor, for each of the plurality of ambient RF signals, at least a frequency, a received power, a modulation characteristic, and a temporal arrival characteristic; constructing an ambient RF fingerprint from aggregate characteristics of the plurality of ambient RF signals over a sliding window; comparing the ambient RF fingerprint to a baseline fingerprint; and issuing a reverse sinkhole alert when a deviation between the ambient RF fingerprint and the baseline fingerprint exceeds a threshold and indicates a concentration of wireless traffic toward a single external endpoint not present in the baseline fingerprint.
22. The method of claim 21, further comprising extracting a physical-layer identity or cell identifier from at least one broadcast channel, and detecting an anomaly in which multiple different logical station identifiers are relayed by a same physical transmitter.
23. The method of claim 21, wherein comparing the ambient RF fingerprint to the baseline fingerprint comprises computing a statistical distance over histograms of received power and frequency occupancy.
24. The method of claim 21, further comprising determining at least one of a direction of arrival or an angle of arrival using an antenna array, and detecting that signals associated with multiple logical cells arrive from a common angle.
25. The method of claim 21, wherein comparing the ambient RF fingerprint to the baseline fingerprint includes applying a machine learning model trained on baseline ambient RF data.

26. The method of claim 21, wherein the software-defined radio receiver is a general-purpose Wi-Fi, Bluetooth, or cellular receiver reconfigured by software to collect raw in-phase and quadrature samples.
27. The method of claim 21, further comprising: receiving a sequence of timing reference signals from a candidate gateway; measuring inter-arrival jitter of the timing reference signals; and detecting a gateway jitter sentry condition when the inter-arrival jitter exceeds a threshold or exhibits a periodic structure associated with a software-emulated gateway.
28. A method for gateway jitter sentry detection, comprising: receiving, at a wireless device, a sequence of timing reference signals from a candidate gateway; measuring inter-arrival jitter of the timing reference signals; extracting a jitter pattern across a series of intervals; comparing the jitter pattern to a known clock signature associated with the candidate gateway; and declaring a gateway anomaly when the jitter pattern deviates from the known clock signature or exhibits a signature of a software-emulated gateway.
29. The method of claim 28, further comprising combining the gateway anomaly with a reverse sinkhole alert produced by the method of claim 21 to generate a composite threat score.
30. The method of claim 28, further comprising extracting clock phase noise or packet timestamp quantization as a jitter feature, and identifying a software-defined radio emulation based on timestamp granularity.