On-Chain Multi-Factor Authentication for IoT Access Control

Qian Feng
Fairleigh Dickinson University
qianfeng2021@gmail.com

Yaoqing Liu
Fairleigh Dickinson University
yliu@fdu.edu

Stephanie Schuckers
UNC Charlotte
sschucke@charlotte.edu

Erdal Kose
Fairleigh Dickinson University
ekose@fdu.edu

***Abstract*—The growth of 5G connectivity and the Internet of Things (IoT) has increased both the number of devices that require access control and the value of compromising them. Multi-factor authentication mitigates the weakness of password-only login, but most deployed schemes still rely on an always-on, centralized authentication server that stores credentials and constitutes a single point of failure. This paper presents a multi-factor authentication framework for IoT access control in which the authenticator of record is an Ethereum smart contract. The framework combines three authenticators across two categories: a biometric check (something you are), a wallet signing key held on the user's device (something you have), and a separate hardware security token that produces a time-bound signature (a second, independent something you have). The contract verifies the two possession authenticators and enforces the access-control policy, so no online authentication server is required at verification time. The biometric is deliberately verified off-chain, because biometrics are irrevocable and a public ledger is transparent; it is therefore a client-side gate that the contract does not enforce, and it protects only users of an honest client. We implement a prototype on an Ethereum test network and report per-step latencies together with the measurement method. We state the framework's boundaries explicitly: a trusted registrar, on-chain pseudonymity rather than anonymity, a client-enforced biometric, an OTP that is time-bounded but neither challenge-bound nor single-use, and an access grant that is not yet scoped to a device session. We give a threat model, a per-attack security analysis, an analytic gas-cost estimate, and concrete improvements, including a challenge–response OTP, a verifiable biometric binding, and a session-scoped access grant.**

***Index Terms*—Blockchain, decentralized application, multi-factor authentication, Ethereum, smart contract, access control, IoT security, one-time password.**

I. Introduction

Online services have become the default for work, commerce, and control of physical infrastructure, and the shift accelerated during the COVID-19 period. The same shift has raised the payoff for attackers, so there is a continuing need for authentication that is stronger than a shared secret.

Authentication mechanisms are commonly grouped by how many independent factors they combine. Single-factor authentication (SFA), typically a username/password, is the most deployed and the weakest: passwords are guessable, reused across sites, and exposed in bulk by breaches [1]. Two-factor authentication (2FA) adds a second check, usually an SMS one-time password (OTP), but remains vulnerable to SIM-swapping and code-forwarding; Siadati *et al.* report a forwarding variant that succeeded about half the time [2]. Adding a biometric yields three-factor authentication (3FA). Crucially, a *factor* is one of three *categories—something you know* (a password), *something you have* (a token or key), and *something you are* (a biometric)—not an individual credential: two authenticators drawn from the same category count as a single factor, because they share failure modes [3]. Biometrics cannot be forgotten or casually shared, which makes the inherence category attractive for high-assurance remote access.

Adding factors does not remove the structural weakness of most deployments: a centralized server that stores and verifies credentials. Such a server is a single point of failure [3] and a concentrated target for the biometric templates it holds—and because a biometric cannot be revoked like a password, its capture is especially damaging. Human factors compound this: industry breach data indicate that a large share of breaches involve a human element, such as errors, social engineering, or the use of stolen credentials [4], and social-engineering attacks that trick users into forwarding their own OTPs occur in the wild [2].

We therefore ask whether the verification and storage functions of an authentication system can be moved off a trusted online server and onto a public blockchain, while combining multiple independent authenticators. We present a multi-factor framework built as an Ethereum decentralized application (DApp) in which the *authenticator of record* is the smart contract: it holds only non-sensitive registration data and enforces the on-chain factors and access policy. By *on-chain* we mean where authentication is enforced, not that every check runs on-chain—the biometric is verified off-chain on the user's own device by design, because a biometric is irrevocable and a public ledger is transparent, so its template must never touch the chain. Concretely, the three authenticators are:

- **Something you are:** a biometric (face) check performed locally by the client and *not* verified by the contract;
- **Something you have (1):** the user's Ethereum wallet key, which signs every transaction and gates a deposit-and-whitelist step;
- **Something you have (2):** a separate hardware security token that emits a time-bound signature used as an OTP.

By the definition above these three authenticators span *two* categories, inherence and possession, so, strictly, the scheme is two-factor with an extra independent possession authenticator, not three-factor. Moreover, only the two possession authen-

ticators are enforced by the contract; the biometric raises assurance only when the user's legitimate client is used. We keep the two possession authenticators distinct because they live on physically separate devices with disjoint compromise surfaces, and we discuss in Section IX how the biometric could be bound to the contract and why a genuine third (knowledge) factor is hard to verify on a public chain.

Contributions. (1) We design a multi-factor IoT access-control framework in which credential storage and the two possession authenticators are verified by a smart contract acting as the authenticator of record, removing the online authentication server as a runtime single point of failure; a client-side biometric check gates entry to the on-chain flow. (2) We give an explicit threat model and a per-attack security analysis, and we state plainly which attacks the design does and does not address. (3) We implement a working prototype (face-api.js verifier, Solidity contract, web3.js front end, MetaMask wallet), report per-step latency with its measurement method, and give an analytic estimate of gas cost. (4) We identify the framework's principal limitations—owner trust, a client-enforced biometric, two possession authenticators in a single category, an OTP that is time-bound but not challenge-bound, an access grant that is not session-scoped, and unmeasured gas cost—and propose concrete remedies. We do *not* claim the system is trustless or anonymous: a device owner is trusted to register users and set policy and learns the identity-to-wallet mapping. The design removes the always-on credential server, not every trusted party.

II. Background

Our framework is built on Ethereum. We summarize only the properties needed later.

A. Blockchain properties

A blockchain is a distributed, append-only ledger [5], [6]. Four properties matter here: it is *immutable* (an included transaction cannot be altered without redoing later work and out-voting the network); *replicated*, avoiding a single point of failure; based on *public-key cryptography*, so users transact under key pairs; and governed by a *consensus mechanism* (PoW, PoS, PBFT) that lets distrusting nodes agree on one ledger state despite Byzantine faults [7], [8]. Note that immutability protects the *history* of transactions; the current state of a contract can still change through any function the contract exposes. Accounts are pseudonymous rather than anonymous, since all transactions of an address are publicly linkable [9], [10]. These properties underpin applications in voting [11], cloud security [12], energy trading [10], research-data rights management [13], and document version control [14].

B. ECDSA on Ethereum

Ethereum accounts are secured with the Elliptic Curve Digital Signature Algorithm (ECDSA): a private key signs a message, and anyone can recover the corresponding address—the same primitive we reuse for the OTP [15]. Its compact keys keep on-chain verification cheap. One caveat: ECDSA is only as strong as its nonces, since biased or reused nonces leak keys [16], which is relevant to any signing device, including our security token.

C. Ethereum, smart contracts, and DApps

Ethereum is a programmable blockchain [17]. Externally owned accounts are controlled by private keys and contract accounts by on-chain code; each account carries a nonce, balance, code hash, and storage root [18]. A *wallet* (e.g., MetaMask) manages a user's key and signs on their behalf without custodying funds [19]. A *smart contract* is EVM code that runs identically on every node and enforces its rules without an intermediary [20]; once deployed, its code and interactions are public at its address. Contracts are written in Solidity and compiled in Remix. A *decentralized application (DApp)* pairs a front end with a smart-contract back end [21], [22], so every change to stored data is recorded on the ledger and users need not reveal real identities [19]. Every EVM operation costs *gas*, which is central to our cost discussion (Sections VIII and IX).

III. Related Work

Several works place part of the authentication logic on smart contracts; the decentralized-identity literature addresses richer identity but not on-chain enforcement of multiple authentication categories. We compare against the three closest systems.

Amrutiya *et al.* [23] propose Ethereum-based 2FA that combines a conventional password (knowledge) with an OTP (possession): the user stores an OTP hash on-chain and an auxiliary OpenSSH server helps the contract verify it, removing SMS/email delivery but adding an external server. Their scheme is therefore already cross-category, pairing knowledge with possession. Oktian *et al.* [24] propose blockchain 2FA for web services in which the user signs a server-generated random challenge with a private key before login; the challenge design is disposable and resists replay. Homoliak *et al.* [25] present SmartOTPs, an air-gapped 2FA smart-contract wallet combining a private key with OTPs derived from Merkle trees and hash chains, targeting self-custody of crypto-tokens; both of its authenticators are possession-based. Outside the blockchain setting, FIDO2/WebAuthn [26] provides phishing-resistant, challenge-bound hardware-backed authentication, but relies on a relying-party server to issue challenges and store credentials.

Relative to these, our framework does not introduce the first cross-category on-chain scheme; its difference is the *category* it adds. It pairs two possession authenticators that the contract verifies without an auxiliary server with an *inherence* (biometric) authenticator, targeting IoT access control rather than web login or wallet custody. Because our biometric is enforced by the client rather than by the contract, this difference holds only under an honest client (Section VII). We also note that the challenge–response designs of Oktian *et al.* and of WebAuthn are stronger than our current time-bound OTP against replay; we adopt the idea as the basis for the improvement proposed in Section IX.

IV. Threat Model and Design Goals

We make the trust and adversary assumptions explicit, because they determine which of the framework's claims hold. The protected asset is access to an IoT device. Sensitive credentials are the user's biometric template, wallet signing key, and security-token key.

A. Trusted parties and components

- **Device owner (trusted registrar).** The owner deploys the contract, vets and registers users, sets the access policy, and holds the identity→wallet mapping. The owner is assumed honest.
- **User's personal device and client.** Holds the wallet key and runs the DApp, including the off-chain biometric verifier; trusted unless the device is compromised or stolen. In particular, the biometric is enforced only if the user interacts through this unmodified client.
- **Security token.** A separate device holding the token key; trusted unless physically stolen.
- **Ethereum network.** Assumed to provide standard blockchain guarantees; we do not consider adversaries who control consensus.

B. Adversary capabilities

The adversary may (i) know or steal a user's password (motivating factors beyond knowledge), (ii) observe and inject messages on the public network, including reading pending transactions in the mempool, (iii) attempt to compromise any *one* authenticator, and (iv) mount social-engineering attacks. The adversary cannot break ECDSA or keccak256, cannot compromise the owner, and cannot simultaneously steal both the user's personal device and the security token.

C. Design goals

G1: no online, always-on authentication server storing credentials. G2: multiple independent authenticators spanning at least two categories, such that compromise of one authenticator does not grant access. The contract enforces G2 for the two possession authenticators; the biometric contributes to G2 only against an adversary who goes through the legitimate client. G3: no sensitive data (biometric templates, private keys) written on-chain. G4: owner-defined access-control policy (whitelist, time window) enforced by the contract. The following are *not* addressed by the current design and are treated as limitations in Section IX: securing the channel between the contract and the IoT device and scoping the access grant to a device session; a malicious client, or a direct contract call, that bypasses the off-chain biometric check; reuse of the OTP within its validity window; and owner compromise.

V. Framework Overview

Fig. 1 shows the three component groups: (i) user and security token, (ii) device owner and IoT device, and (iii) the DApp, whose back end is the smart contract and whose off-chain helper is the AI (biometric) verifier.

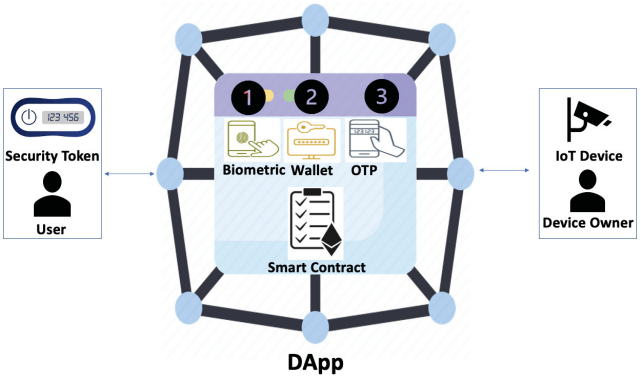


Fig. 1. General framework of the proposed on-chain multi-factor authentication DApp

A. Components

User and security token. A user is any party seeking access. The security token is a private hardware device that, when connected to obtain Unix epoch time, emits a time-bound signature (our OTP) once per 100-second window, synchronized to the contract's clock.

IoT device and owner. The device requires high-assurance access control (e.g., a security door or camera). The owner owns the device and the contract, registers users, and sets policy.

DApp. The front end lets users interact with the contract directly; the contract stores non-sensitive registration data and enforces the on-chain factors. The three authenticators are: (1) an off-chain biometric check, kept off-chain precisely because on-chain data is public and a biometric is irrevocable; (2) a wallet step in which the user proves key possession and satisfies the policy; and (3) an OTP step proving possession of the security token.

B. Authentication lifecycle

The lifecycle has five stages, shown as a sequence diagram in Fig. 2; the numbers in parentheses below refer to the messages in that figure.

Step 1 — Construction and registration. The owner deploys the contract (1.1), receives the user's token public key and face image (1.2), verifies identity, generates a wallet for the trusted user, records the token public key and wallet address on-chain (1.3), and sets the access policy (1.4). No biometric or private key is stored on-chain.

Step 2 — Biometric (off-chain). The user re-captures their biometric (2.1); the client-side verifier compares it to the registered reference (2.2) and, on success, unlocks the DApp's on-chain steps (2.3), otherwise the user retries (2.4). The contract receives no evidence of this step.

Step 3 — Wallet (on-chain). The user signs a deposit transaction (3.1), and the contract checks the access window and the whitelist (3.2). If the deposit is sent outside the owner-defined access window, the contract *rejects* it (3.3): the transaction reverts and the Ether is not transferred (only the gas

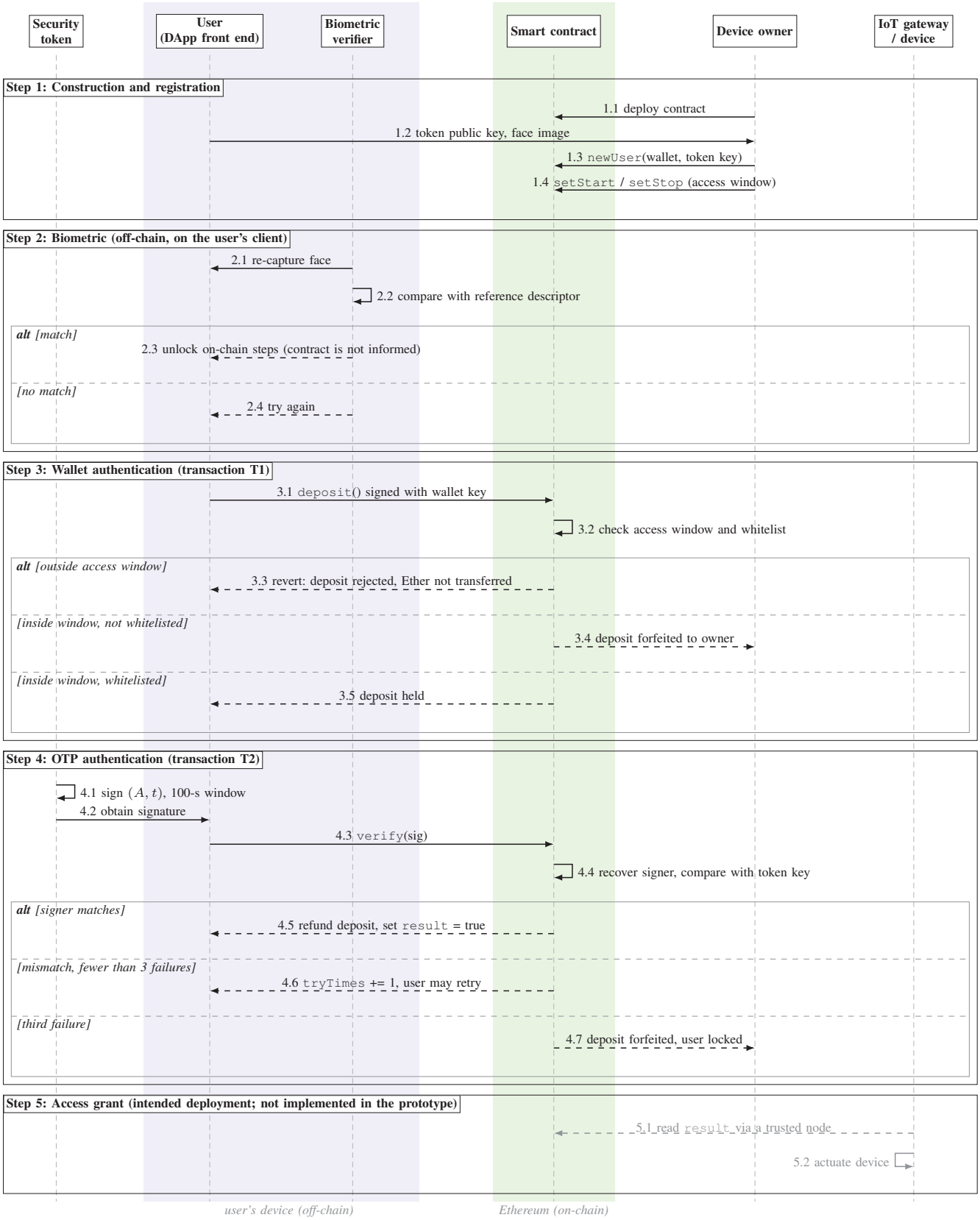


Fig. 2. Authentication lifecycle. Shaded columns separate what runs on the user's device (off-chain) from what the smart contract enforces (on-chain). Step 5 (grey) is the intended device-side path and was not implemented in the prototype.

fee is spent). Inside the window, a deposit from a whitelisted address is held by the contract (3.5), whereas a deposit from a non-whitelisted address is *forfeited* to the owner (3.4). Signing is itself the proof of key possession, and the deposit adds an economic disincentive to unauthorized attempts (we revisit its calibration in Section IX).

Step 4 — OTP (on-chain verification). The token signs a message over its current 100-second window and the contract address (4.1); the user obtains the signature (4.2) and submits it (4.3). The contract reconstructs the message for the current window, recovers the signer, and compares it to the token key registered for the caller (4.4). On success it refunds the deposit and records the outcome (4.5). Each failure increments a per-user counter (4.6); when the counter reaches three, the deposit is forfeited and further attempts are refused (4.7). In the prototype this counter is cumulative—a successful login does not reset it—so the lock is triggered by three failed attempts in total rather than three consecutive ones, and is lifted only when the owner re-registers the user (`newUser` resets the counter).

Step 5 — Access grant. A successful verification sets the user's `result` flag to true; this flag is the access grant. In the intended deployment, the IoT device, or an edge gateway in front of it, reads this state from an Ethereum node it trusts (5.1) and actuates on a true value (5.2); the DApp never talks to the device directly, so the security of this path rests on the device's view of the chain and on its own actuation logic. The prototype ends at the on-chain grant: device actuation was not implemented, and the flag is neither scoped to a device request, time-limited, nor reset after use. A device therefore cannot distinguish a fresh grant from a stale one; we treat this as a limitation and propose a session-scoped grant in Section IX.

VI. Implementation

We built a decentralized web application. The biometric verifier uses face-api.js (0.22.2) [27], which detects a face, extracts a 128-dimensional descriptor with its pre-trained recognition network, and accepts a probe whose Euclidean distance to the user's reference descriptor is below a threshold. The reference descriptor is computed from enrollment images of the user, and the threshold is 0.6 by default. The network itself is not retrained per user. No liveness or presentation-attack detection is applied, and we did not measure false-acceptance or false-rejection rates; the accuracy reported for the underlying model on public benchmarks [27] does not transfer to our setting, so we make no quantitative claim about the biometric. The smart contract is Solidity (0.7.6), written and compiled in Remix. The front end is HTML/CSS/JavaScript connected to the contract via web3.js, with MetaMask (9.5.5) as the wallet. The security token was emulated on a separate machine. We deployed and tested on the Ethereum Kovan test network; the user wallets in our experiments were generated with Ganache.[1] As noted in Step 5, no physical IoT device was connected; the prototype demonstrates authentication and the on-chain access grant, not actuation.

[1]Kovan has since been deprecated; a current testnet (e.g., Sepolia or Hoodi) should be used for future measurements. This does not affect the design.

```
struct User {
   string publicKey; // token public key (address
       form)
   bool isUser; // whitelist membership
   uint balance; // held deposit
   uint tryTimes; // failed OTP attempts (cumulative
       )
   bool result; // last OTP outcome (access grant)
}
mapping(address => User) user;
address payable owner;

modifier isOwner() {
   require(msg.sender == owner, "not owner");
   _;
}
```

Algorithm 1. User record and ownership modifier

```
function newUser(address w, string memory pk) public
     isOwner {
   user[w] = User(pk, true, 0, 0, false);
}
function deleteUser(address w) public isOwner {
   user[w].isUser = false;
}
function updateUser(address w, string memory pk)
    public isOwner {
   user[w].publicKey = pk;
}
function setStart(uint t) public isOwner {
    todayStart = t; }
function setStop(uint t) public isOwner { todayStop
    = t; }
```

Algorithm 2. Owner management (CRUD) and access-time policy

A. Construction

The verifier is embedded in the DApp: after obtaining a user's face image the owner computes the user's reference descriptor and adds it to that user's DApp instance. The contract is compiled and deployed to obtain its ABI and address; the owner connects the DApp to the contract via web3.js using these two parameters. Because the ABI/address pair is unique per deployment, each device/owner/user triple has its own DApp instance.

B. Registration

The user supplies only two items: a face image and the token's public key. Once the owner trusts the user, the owner generates an Ethereum wallet (a 256-bit private key with an ECDSA-derived public key) and records the user's non-sensitive fields on-chain. Algorithm 1 shows the user record and ownership modifier; Algorithm 2 shows the owner's management and policy functions.

C. Wallet authentication

Client side. Every DApp action creates a transaction that MetaMask signs with the user's key; a valid signature proves

```
receive() external payable {}

modifier inHours() {
   require(todayStart <= block.timestamp
   && block.timestamp <= todayStop, "outside hours")
       ;
   _;
}
modifier inWhitelist(address a) {
   require(user[a].isUser, "not whitelisted");
   _;
}
modifier funded(address a) {
   require(user[a].balance >= 1 ether, "not funded")
       ;
   _;
}
function deposit() external payable inHours {
   user[msg.sender].balance += msg.value;
   if (!user[msg.sender].isUser) {
      transToOwner(msg.sender); // forfeit
   }
}
function transToOwner(address a) internal {
   uint amt = user[a].balance;
   user[a].balance = 0;
   owner.transfer(amt);
}
```

Algorithm 3. Wallet authentication (deposit, window, whitelist)

key possession. To proceed, the user signs a `deposit` transaction.

Contract side. Algorithm 3 accepts Ether via `receive`. The `inHours` modifier reverts any deposit sent outside the access window, so such a deposit is rejected rather than forfeited. Within the window, the deposit is credited to the sender, and if the sender is not whitelisted it is immediately transferred to the owner. The minimum deposit of 1 ether is not checked here but at verification time (`funded`).

D. OTP authentication

Signing format. Let A be the contract address and $t = T - (T \bmod 100)$ the start of the current 100-second window, where T is the Unix time in seconds. The token computes $h = \mathrm{keccak256}(A \,\|\, t)$, with A encoded as 20 bytes and t as a 32-byte unsigned integer (Solidity `abi.encodePacked`), and signs it as an Ethereum personal message following EIP-191 [28], i.e., it produces an ECDSA signature (r, s, v) over

$$\mathrm{keccak256}\big(\texttt{\textbackslash x19Ethereum Signed Message:\textbackslash n32} \,\|\, h\big).$$

The user pastes the 65-byte signature into the DApp. Generation is off-chain and can be offline, so the token key never leaves the device.

Verification (contract). The contract reconstructs the same digest for the current window (Algorithm 4), recovers the signer, and compares it to the token key registered for `msg.sender` (Algorithm 5). Quantizing the timestamp to the 100-second window ensures the token and contract derive the same message within a window. Two properties of the prototype code should be stated precisely. First, the contract address enters the digest through the caller-supplied parameter `to`, not through `address(this)`; the contract therefore does not itself enforce that the signature was made for this contract. Replacing `to` with `address(this)` is a one-line fix that we adopt for future versions. Second, the window is evaluated at block-inclusion time, so a signature produced near the end of a window may be checked in the next window and counted as a failure; accepting the previous window as well would remove these spurious failures at the cost of a longer validity period.

```
function getMessageHash(address to) public view
    returns (bytes32) {
   uint t = block.timestamp - (block.timestamp %
       100); // 100s window
   return keccak256(abi.encodePacked(to, t));
}
function getEthSignedMessageHash(bytes32 h) public
    pure returns (bytes32) {
   return keccak256(abi.encodePacked(
   "\x19Ethereum Signed Message:\n32", h));
}
```

Algorithm 4. OTP message reconstruction (windowed)

```
function verify(address to, bytes memory sig)
external inHours inWhitelist(msg.sender) funded(msg.
    sender) {
   require(user[msg.sender].tryTimes < 3, "locked");
   bytes32 h = getMessageHash(to);
   bytes32 eh = getEthSignedMessageHash(h);
   address signer = recoverSigner(eh, sig);
   if (signer == parseAddr(user[msg.sender].
       publicKey)) {
      user[msg.sender].result = true;
      refund(msg.sender);
   } else {
      user[msg.sender].tryTimes += 1;
      if (user[msg.sender].tryTimes >= 3) {
         transToOwner(msg.sender);
      }
   }
}
function refund(address a) internal funded(a) {
   uint amt = user[a].balance;
   user[a].balance = 0;
   payable(a).transfer(amt);
}
function recoverSigner(bytes32 eh, bytes memory sig)
internal pure returns (address) {
   (bytes32 r, bytes32 s, uint8 v) = splitSignature(
       sig);
   return ecrecover(eh, v, r, s);
}
```

Algorithm 5. OTP verification, refund, and signer recovery

If the recovered signer matches the registered token key, the caller has proven possession of both the wallet key and the token within the current window; the deposit is refunded and the access grant (`result`) is set.

VII. Security Analysis

We revisit the threat model of Section IV per attack. The claims below hold only within that model.

Attacks the design addresses. Registration data and on-chain verification live on a replicated ledger, so there is no

online credential store to breach or take offline (G1, G3). No password is a factor, so a leaked password is useless without the two possession credentials. The OTP is never sent over SMS or forwarded to the user, so the SMS-based attacks of [2] do not apply.

Integrity of the whitelist and token keys (auditable, not immutable). The whitelist and the registered token keys can be changed only by the owner account (Algorithm 2), and every change is recorded on the ledger. Blockchain immutability therefore gives a tamper-evident, auditable history of the security state; it does not prevent a malicious or compromised owner from deleting users or replacing keys.

Single-authenticator compromise (partially addressed, G2). Losing the wallet key alone, or the token alone, is insufficient: the contract still requires the other possession credential. The biometric adds protection only if the attacker uses the legitimate client; an attacker who holds both possession credentials can call the contract directly and is not stopped by it. The residual weakness is category overlap—two authenticators are possession-based—so stealing both of the user's devices in one theft defeats everything the contract checks.

OTP reuse (not fully addressed). The OTP signs `(contract address, time window)`; it is not bound to a fresh challenge, is not consumed after use, and is submitted in a public transaction. Because `verify` compares the recovered signer with the token key registered for `msg.sender`, a network observer who sees the signature in the mempool cannot use it from their own address. However, an attacker who has stolen the victim's wallet key can resubmit an observed signature from the victim's address for the rest of the 100-second window without possessing the token, provided the access window and deposit conditions are met. The window thus bounds, but does not eliminate, reuse; it reduces the token to a time-limited authenticator whenever the wallet key is compromised. This is the most important open weakness and motivates the redesign in Section IX.

Biometric bypass (not addressed). The biometric is verified off-chain with no cryptographic binding to the on-chain steps, so a compromised client, or any party calling the contract directly, can skip or spoof it. Without liveness detection the check is also open to presentation attacks such as printed photographs. The biometric should therefore be read as a usability-level gate on the honest client, not as an enforced factor.

Owner compromise / contract–device path (out of scope). A malicious owner can register rogue users or, having generated the wallet, retain the user's key; and the path from a successful authentication to the physical device, including grant freshness, is not secured here. Both are explicit trust assumptions, not defended attacks.

VIII. Evaluation

Setup and method. The DApp used MetaMask to connect to Kovan. Each row of Table I is the wall-clock time from the completion of the previous stage (or the user's action that starts the step) until the DApp observed the step's outcome. For steps that submit a transaction, the interval ends when the transaction receipt is returned, so it includes signing, propagation, block inclusion, and one confirmation, not only EVM execution. Each value is the mean of 5 runs. The column "Tx" maps each row to the transaction it belongs to: rows marked T1 are stages of the single `deposit` transaction and rows marked T2 are stages of the single `verify` transaction, observed by the front end; they are not separate transactions. "Forfeit" rows are alternative outcomes of the same transaction and are not on the success path.

TABLE I. Prototype step latencies (Ethereum Kovan test network)

Process	Role	Attribute	Tx	Time (s)
Construction	Owner	Deploy contract	own	∼3
Construction	Owner	Add user to whitelist	own	∼5
Construction	Owner	Set available hours	own (2)	∼10
Biometric	User	Re-capture	off	∼2
Biometric	Verifier	Verify	off	∼3
Wallet	User	Deposit	T1	∼3
Wallet	Contract	Check conditions	T1	∼12
Wallet	Contract	Deposit succeed	T1	∼5
Wallet	Contract	Forfeit deposit	T1	∼5
OTP	Token	Generate OTP	off	∼3
OTP	User	Obtain OTP	off	<3
OTP	User	Input OTP	off	∼4
OTP	Contract	Get OTP	T2	<2
OTP	Contract	Verify OTP	T2	∼14
OTP	Contract	Refund deposit	T2	∼5
OTP	Contract	Forfeit deposit	T2	∼5

off: off-chain; own: own transaction, (2): two transactions;
T1: `deposit` transaction; T2: `verify` transaction.

Results. Summing the success-path rows gives about 56 s, i.e., a user completes authentication in roughly one minute without optimization, of which about 5 s is the off-chain biometric and most of the remainder is waiting for the two transactions to be included. An owner completes construction in about 20 s. These figures indicate feasibility for interactive access where a delay of this order is acceptable, such as opening a door.

Analytic gas estimate. We did not measure gas in the prototype; the following is an estimate from the EVM fee schedule [18], not a measurement. A successful authentication requires two user transactions (`deposit` and `verify`), each paying the 21,000-gas intrinsic cost. `deposit` adds one storage update of the balance. `verify` adds signer recovery (the `ecrecover` precompile costs 3,000 gas), reading and parsing the stored key string, storage updates of `result` and `balance`, and an Ether transfer. Each authentication is therefore expected to cost on the order of 10^5 gas, and storage grows by a constant-size record per registered user. Storing the token key as an `address` rather than a string would remove the parsing cost.

What these numbers do and do not show. The latencies are dominated by block-inclusion time on a now-deprecated testnet and should be read as feasibility evidence, not a benchmark. The evaluation does not include measured gas cost, fiat/ETH cost, latency percentiles, behavior under network congestion or concurrent users, or a quantitative comparison

with centralized MFA. These are the first items of future work (Section IX).

IX. Discussion, Limitations, and Future Work

We consolidate the limitations surfaced above with a concrete path for each.

L1: Cost, latency, and evaluation scope. On-chain storage and verification cost gas, and Ether prices are volatile, so per-access cost can be significant. The next step is to *measure* gas per operation on a current testnet (Sepolia or Hoodi), report repeated-run latency distributions, and stress-test with concurrent users. A centralized TOTP or WebAuthn [26] server verifies in milliseconds at negligible marginal cost; our scheme trades that latency and a per-access fee for removing the online credential server, and a quantitative side-by-side comparison under varying load is planned. Because latency is dominated by block inclusion, deploying the contract on a Layer-2 rollup, whose block intervals are much shorter than the 12-second slots of the Ethereum main chain, should reduce both latency and fees substantially; we intend to compare the transaction lifecycle on such networks.

L2: OTP reuse to challenge–response (proposed, not implemented). We would replace the time-window OTP with a challenge–response flow after [24]: the contract issues a fresh single-use nonce, the token signs a typed message over `(nonce, msg.sender, chain ID, address(this), action)`, e.g., following EIP-712 [29], and the contract verifies and *consumes* the nonce, never accepting a bare timestamp signature. This removes the reuse window, binds the OTP to the caller, the chain, the contract, and the requested action, and fixes the contract-binding gap noted in Section VI.

L3: Client-enforced biometric. Binding the biometric outcome to the on-chain flow, e.g., a TEE-attested (Trusted Execution Environment) or zero-knowledge assertion the contract can check, would stop a compromised client or a direct contract call from skipping factor 1. A deployable biometric would also need liveness detection and an evaluation of false-acceptance and false-rejection rates on a stated dataset. Until then we state the honest-client assumption openly.

L4: Session-scoped access grant. The Boolean `result` should be replaced by a grant that a device can consume exactly once: on success the contract would emit an event carrying the user, a device identifier, a session identifier, and an expiry time, and the device or gateway would accept a session only before its expiry and only once, after which the grant is cleared. Securing the device side (mutual authentication between gateway and device, and an encrypted channel) is required before deployment and is complementary to this work.

L5: Deposit and lockout policy. The fixed 1-ether deposit suits a test network but not production, where the Ether price is volatile and failed attempts caused by clock skew or token errors should not cost a legitimate user the full amount. A production contract would make the deposit an owner-set parameter calibrated to the value of the protected device, could denominate it in a stablecoin token rather than in Ether, and would reset the failure counter on success so that the lock reflects consecutive failures.

L6: Two possession authenticators / adding a knowledge factor. The scheme spans only inherence and possession. A local PIN that merely unlocks the wallet or token does *not* add a category: the contract still sees only a signature, so it verifies possession and "the user knew a PIN" is invisible to it. A genuine, contract-verified knowledge factor must prove a secret *without revealing it*: a plaintext PIN is public in the transaction and a stored hash is offline-guessable, so it requires a PAKE (Password-Authenticated Key Exchange) or a zero-knowledge proof of knowledge, which is nontrivial on a public chain. Separately, letting the user generate the key, with the owner recording only the public key, removes the owner's transient key custody and is a drop-in registration change.

L7: Owner as trusted registrar. The owner is a trust anchor; threshold/multi-owner or self-sovereign enrollment would broaden the "decentralized" claim.

X. Conclusion

We presented an on-chain multi-factor authentication framework for IoT access control, in which an Ethereum smart contract is the authenticator of record: it enforces two possession-based authenticators, while an off-chain biometric gates entry to that flow on the user's client. The design removes the always-on centralized authentication server as a runtime single point of failure, keeps sensitive data off-chain, and lets a device owner enforce whitelist and time-window policies on-chain; a prototype completes authentication in about a minute on a public testnet. We were deliberate about scope: the three authenticators span two categories (inherence and possession) rather than three; only the possession authenticators are contract-enforced; the system relies on a trusted owner and offers pseudonymity rather than anonymity; the OTP is time-bound but not challenge-bound or single-use; and the access grant is not yet scoped to a device session. The next steps are a challenge–response OTP bound to the caller, chain, and contract; a verifiable biometric binding; a session-scoped grant with device-side enforcement; measured gas cost and latency on current networks, including Layer-2 rollups; and reduced reliance on the owner. We believe the direction, moving authentication storage and enforcement onto a public ledger while combining independent authenticators, is a useful building block for high-assurance IoT access control.

References

[1] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.

[2] H. Siadati, T. Nguyen, P. Gupta, M. Jakobsson, and N. Memon, "Mind your SMSes: mitigating social engineering in second factor authentication," *Comput. Secur.*, vol. 65, pp. 14–28, Mar. 2017.

[3] A. Ometov, S. Bezzateev, N. Mäkitalo, S. Andreev, T. Mikkonen, and Y. Koucheryavy, "Multi-factor authentication: a survey," *Cryptography*, vol. 2, no. 1, Art. no. 1, 2018.

[4] Verizon, "2024 Data Breach Investigations Report," Verizon Business, 2024. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[5] S. Nakamoto, “Bitcoin: a peer-to-peer electronic cash system,” 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. Sebastopol, CA, USA: O’Reilly Media, 2014.

[7] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: architecture, consensus, and future trends,” in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, 2017, pp. 557–564.

[8] L. Lamport, R. Shostak, and M. Pease, “The Byzantine generals problem,” *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[9] E. Heilman, F. Baldimtsi, and S. Goldberg, “Blindly signed contracts: anonymous on-blockchain and off-blockchain bitcoin transactions,” in *Financial Cryptography and Data Security (FC 2016 Workshops)*, ser. LNCS, vol. 9604. Berlin, Germany: Springer, 2016, pp. 43–60.

[10] N. Z. Aitzhan and D. Svetinovic, “Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams,” *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 840–852, Sep.–Oct. 2018.

[11] D. Khoury, E. F. Kfoury, A. Kassem, and H. Harb, “Decentralized voting platform based on Ethereum blockchain,” in *Proc. IEEE Int. Multidisciplinary Conf. Eng. Technol. (IMCET)*, 2018, pp. 1–6.

[12] J. H. Park and J. H. Park, “Blockchain security in cloud computing: use cases, challenges, and solutions,” *Symmetry*, vol. 9, no. 8, Art. no. 164, 2017.

[13] A. T. Panescu and V. Manta, “Smart contracts for research data rights management over the Ethereum blockchain network,” *Sci. Technol. Libraries*, vol. 37, no. 3, pp. 235–245, 2018.

[14] N. Nizamuddin, K. Salah, M. A. Azad, J. Arshad, and M. H. Rehman, “Decentralized document version control using Ethereum blockchain and IPFS,” *Comput. Electr. Eng.*, vol. 76, pp. 183–197, Jun. 2019.

[15] G. Nyame, Z. Qin, K.-B. Obour Agyekum, and E. B. Sifah, “An ECDSA approach to access control in knowledge management systems using blockchain,” *Information*, vol. 11, no. 2, Art. no. 111, 2020.

[16] J. Breitner and N. Heninger, “Biased nonce sense: lattice attacks against weak ECDSA signatures in cryptocurrencies,” Cryptology ePrint Archive, Rep. 2019/023, 2019. [Online]. Available: https://eprint.iacr.org/2019/023

[17] D. Vujicic, D. Jagodic, and S. Randic, “Blockchain technology, Bitcoin, and Ethereum: a brief overview,” in *Proc. 17th Int. Symp. INFOTEH-JAHORINA*, 2018, pp. 1–6.

[18] G. Wood, “Ethereum: a secure decentralised generalised transaction ledger,” Ethereum Project Yellow Paper, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[19] Ethereum Foundation, “Ethereum development documentation.” [Online]. Available: https://ethereum.org/en/developers/docs/ (accessed Oct. 2026).

[20] M. Pilkington, “Blockchain technology: principles and applications,” in *Research Handbook on Digital Transformations*, F. X. Olleros and M. Zhegu, Eds. Cheltenham, U.K.: Edward Elgar, 2016, pp. 225–253.

[21] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, “A survey on the security of blockchain systems,” *Future Gener. Comput. Syst.*, vol. 107, pp. 841–853, Jun. 2020.

[22] B. Singhal, G. Dhameja, and P. S. Panda, *Beginning Blockchain: A Beginner’s Guide to Building Blockchain Solutions*. Berkeley, CA, USA: Apress, 2018.

[23] V. Amrutiya, S. Jhamb, P. Priyadarshi, and A. Bhatia, “Trustless two-factor authentication using smart contracts in blockchains,” in *Proc. Int. Conf. Inf. Networking (ICOIN)*, 2019, pp. 66–71.

[24] Y. E. Oktian, S.-G. Lee, and H.-J. Lee, “TwoChain: leveraging blockchain and smart contract for two factor authentication,” in *Proc. 3rd Int. Seminar Res. Inf. Technol. Intell. Syst. (ISRITI)*, 2020, pp. 187–191.

[25] I. Homoliak, D. Breitenbacher, O. Hujnak, P. Hartel, A. Binder, and P. Szalachowski, “SmartOTPs: an air-gapped 2-factor authentication for smart-contract wallets,” in *Proc. 2nd ACM Conf. Advances in Financial Technologies (AFT)*, 2020, pp. 145–162.

[26] W3C, “Web Authentication: an API for accessing public key credentials, Level 2,” W3C Recommendation, Apr. 2021. [Online]. Available: https://www.w3.org/TR/webauthn-2/

[27] V. Mühler, “face-api.js: JavaScript API for face detection and face recognition in the browser,” GitHub repository. [Online]. Available: https://github.com/justadudewhohacks/face-api.js (accessed Oct. 2026).

[28] M. H. Swende and N. Johnson, “EIP-191: Signed data standard,” Ethereum Improvement Proposals, 2016. [Online]. Available: https://eips.ethereum.org/EIPS/eip-191

[29] R. Bloemen, L. Logvinov, and J. Evans, “EIP-712: Typed structured data hashing and signing,” Ethereum Improvement Proposals, 2017. [Online]. Available: https://eips.ethereum.org/EIPS/eip-712