TrustedAP: Using the Ethereum Blockchain to Mitigate the Evil Twin Attack

Phil Fox
pcfox@buffalo.edu
SUNY University at Buffalo
Buffalo, New York

ABSTRACT

The wireless network Evil Twin Attack (ETA) is still only partially mitigated decades after the advent of wireless networking. The ETA is a spoofing attack wherein a malicious actor creates a convincing copy or alternative to a wireless Access Point (AP) to which unsuspecting victims connect for internet access, often in public venues. We introduce TrustedAP: A web and network composite application that provides means for clients to verify AP ownership as trustworthy using minimal (if any) connected time[6]. Ethereum blockchain smart contracts execute code establishing a 'trust side-channel.' This side-channel maintains verification mechanisms that support off-chain messaging between client endpoints and APs. End Users compare message contents with blockchain transcriptions yielding a very strong reason to believe that their connection is trustworthy, genuine, and under positive control of intended public WiFi providers.

CCS CONCEPTS

• **Networks** → **Network properties**; • **Security and privacy** → *Intrusion/anomaly detection and malware mitigation*; • **Systems security** → Operating systems security.

KEYWORDS

evil twin attack, wireless network security, trust side-channel

ACM Reference Format:
Phil Fox. 2021. TrustedAP: Using the Ethereum Blockchain to Mitigate the Evil Twin Attack. In *SRC '21: ACM Student Research Competition, June 2021, New York, NY*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3439695

1 BACKGROUND

1.1 ETA methodology

The ETA is a low-to-medium-skill wireless network Computer Network Attack (CNA) wherein a malicious threat agent (attacker) uses hardware masquerading as (i.e., spoofing) an intended Access Point (AP) for unsuspecting end-users' client devices. Attackers assign an identical or compelling Service Set Identifier (SSID) to a rogue AP and match login credentials where applicable. Unsuspecting end users are prompted to chose a list of available APs using SSID information alone in less-familiar environments (e.g., pubic WiFi furnished airports, restaurants, hotels, etc).

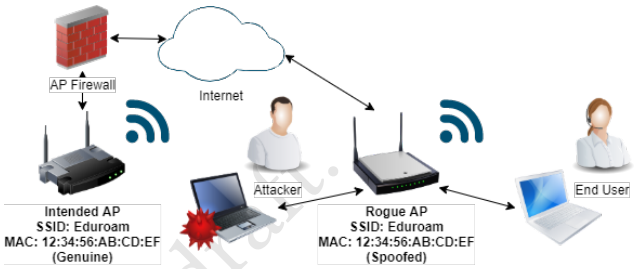

Figure 1: ETA diagram

An ETA is successful when an unsuspecting end user connects to a rogue AP under the attacker's control. ETA success enables attackers' straightforward eavesdropping on victims' network traffic when unencrypted. Further, ETAs leave victims exposed on a deliberately unprotected network facing further vulnerabilities from advanced attacks including malicious captive portals. Today, ETAs are *partially* mitigated by ensuring that TLS/SSL connections or full tunnel Virtual Private Networks (VPNs) protect all client endpoint traffic, but the above vulnerabilities persist in even a temporary lapse of strong web or network security.

Historically, The ETA was particularly effective when default operating system wireless network configurations stored SSIDs and automatically connected when a saved SSID was found within range (e.g., pre-Windows XP SP1). Public and device-default WiFi SSIDs (e.g., "CoffeeShop", "linksys") facilitated ETAs due to their ubiquity as 'remembered' connections across notebook devices. Today, clients often confirm public networks per-use prior to connection, but the lingering issue stands that plaintext, easily duplicated SSIDs give clients a near-zero basis for trust. The advent of mobile hotspotting lends rogue APs a transparent upstream gateway for victims to browse and operate with seemingly normally while connected to a rogue device (illustrated as the AP-to-internet connection in Fig 1. In-short, the *human-factor* basis of connecting to 'a good-looking, working AP' underwrites 2021 ETAs despite the growing protective strength of subsequent WPA and OS generations.

1.2 Related Work

Proposals for mitigating ETAs also subscribe to a trust side-channel methodology in distinct ways. A wide classification of approaches includes establishing *off-network* trust from client to AP that confirms that a network connection between each is trustworthy. A second class of proposals include establishing a trust side-channel

ACM SRC '21, June 2021, New York, NY

https://doi.org/10.1145/3408877.3439695

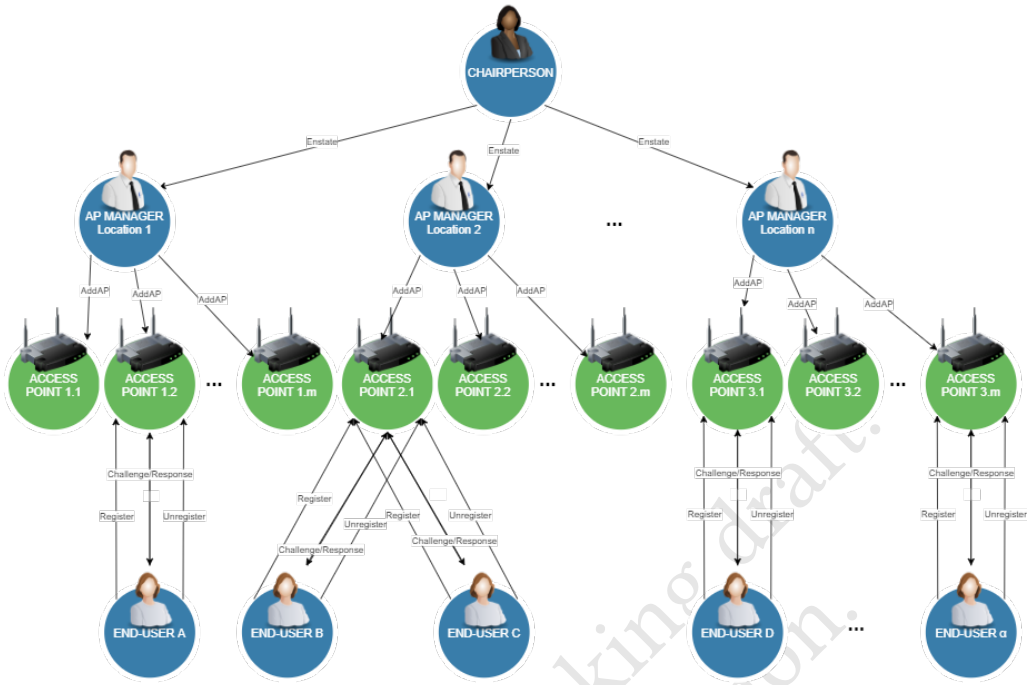

Figure 2: TrustedAP deployment by role

using *on-network* or network-adjacent artifacts which, again, confirm trustworthiness of the network connection between each device.

A paradigm example of an off-network trust side-channel is proposed by Matos et al. [10]. This solution implements a Near Field Communication (NFC) protocol for client endpoints to validate not-yet-trusted APs. A networked device 'pairs' in-proximity to an intended AP or AP proxy hardware. This approach yields nearly optimal 'hands-on' trust considering the extraordinary difficulty in hardware spoofing required for a modified ETA. However, this approach is limited by the physical constraints of NFC connections and the requisite need for non-traditional hardware implementation.

An intuitive approach representing on-network/network-adjacent trust side-channels is proposed by Nakhila et al [12]. This solution extends TLS/SSL certificate validation for APs using similar protocols that facilitate HTTPS secure, trusted communication via the web. This approach yields strong trust while mirroring an existing (web) trust side-channel founded on public and private key cryptographic strength. However, this approach is subject to the same resource requirements as its web counterpart; namely, a need for a *centralized* certificate infrastructure. It remains to be seen how this system would be stood-up or managed.

1.3 Motivation

Despite the effectiveness of above solutions, we have yet to see any widely implemented trust side-channel solution in public WiFi provisioned spaces to-date. Several other effective proposals on mitigating ETAs exist, and each seem to fall within either of the two methodology categories above (further examples include [2, 17]). We also find that each solution faces implementation challenges which are either costly for public WiFi providers and/or fail to meet a threshold of transparency (i.e., ease of use) for widescale End User acceptance/adoption.

TrustedAP is thusly motivated to find a similarly effective trust side-channel solution in a novel way that achieves measures of affordability and end user transparency. TrustedAP is *de facto* novel in that it leverages the Ethereum blockchain for establishing a trust side-channel. However, we aim higher by explicitly considering an ultimate evaluative 'layer': Can TrustedAP achieve a *novel* measure of affordability and transparency that exceeds its predecessors becoming a forefront candidate for adoption? We hypothesize that it can, and we provide a framework for confirming or falsifying such.

2 DESIGN

The overall design for TrustedAP (illustrated in Fig. 3) centers on using the Ethereum blockchain for an AP 'custody chain' and network message record-keeping (on-chain operations). APs and client endpoints launch local applications that record relevant network details using a cryptographic sign/verify protocol while posting/retrieving each others' blockchain records. Should the blockchain records and messages align with the facts of the surrounding network with a sound custody chain, then a connection is trustworthy.

2.1 On-Chain operations

The primary on-chain operations of TrustedAP are role-centric. Trusted AP user/manager roles (illustrated in Figs. 2,3,5) designate Custody Managers, APs, and End Users. End Users connect to APs

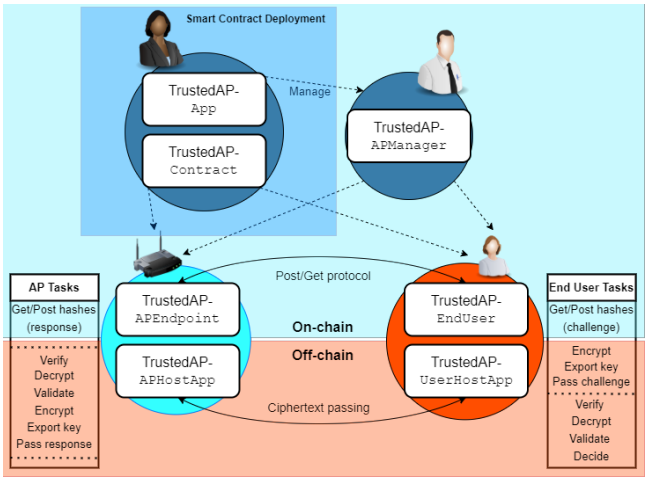

Figure 3: TrustedAP's operational design

with client endpoint devices. For every role, a unique Ethereum address (i.e., account, wallet) and requisite blockchain-facing Elliptical Curve Digital Signature Algorithm (ECDSA) public/private key pair is required. A primary custody manager (chairperson) launches the initial smart contract and designates (`enstates`) second-tier managers who designate (`add`) which APs are trusted. This custody chain is immutable and recorded entirely on the Ethereum blockchain. Conversely, End User roles are self-instantiated by merely creating an address and ECDSA key pair on the Ethereum blockchain. Any AP must disclose its custody chain on any request from an End User. End Users inspect the custody chain and remain connected if sound, or disconnect if unsound (adjudication).

The secondary purpose of Ethereum blockchain implementation is providing verification of messages passed between APs and End User client endpoints. APs/clients merely post Keccak (i.e., SHA3) hash digests of messages passed. Receiving parties verify messages ensuring that there is no discrepancy between AP-client entities communicating on-chain and those that communicate off-chain (i.e., are wirelessly co-networked).

2.2 Off-Chain operations

Primary off-chain operations are summarized by client and AP devices exchanging "network tokens" using RSA public-key signing/verification. Network tokens consist of observed network information from the perspective of client endpoints or APs. Network tokens are non-arbitrary, and are explained further in section 4.2.1. Secondary off-chain operations simply post/retrieve the Keccak hash information from the blockchain and compare messages passed, again, ensuring that each entity is the intended communicator both on and off-chain.

We provide the pseudocode that represents the off-chain interactions in Algs. 1-4.

2.3 Interaction summary

We provide a preliminary 0-level by-step illustration of a normal Trusted-AP interaction series:

Function *challengeAP*
- $\text{NetToken}_c \leftarrow$ ParseNetworkInfo();
- $(CK_{public}, CK_{private}) \leftarrow$ GenerateRSAKeys();
- $\text{VPack}_c \leftarrow$ Enc($CK_{private}$, NetToken_c);
- PostToBlockchain(SHA3(CK_{public}), SHA3(VPack_c));
- SendToAP(CK_{public}, VPack_c);
- Listen();
- **if** *noresponse* **then**
  - Disconnect;
- **else**
  - resolveAP();
- **end**

end

Algorithm 1: Client challenges AP

Function *resolveClient* (CK_{public}, $VPack_c$)
- VerifyOnChain(CK_{public}, VPack_c);
- **if** *mismatch* **then**
  - **break**;
- **end**
- $\text{NetToken}_c \leftarrow$ Dec(CK_{public}, VPack_c);
- Validate(ParseNetworkInfo(),, NetToken_c);
- **if** *mismatch* **then**
  - **break**;
- **else**
  - APRespond();
- **end**

end

Algorithm 2: AP resolves the client device

Function *respondToClient*
- $\text{NetToken}_a \leftarrow$ ParseNetworkInfo();
- $(AK_{public}, AK_{private}) \leftarrow$ GenerateRSAKeys();
- $\text{VPack}_a \leftarrow$ Enc($AK_{private}$, NetToken_a);
- PostToBlockchain(SHA3(AK_{public}), SHA3(VPack_a));
- SendToClient(CK_{public}, VPack_a);

end

Algorithm 3: AP responds to client

Function *resolveAP* (CK_{public}, $VPack_c$)
- VerifyOnChain(AK_{public}, VPack_a);
- **if** *mismatch* **then**
  - Disconnect();
- **end**
- $\text{NetToken}_a \leftarrow$ Dec(AK_{public}, VPack_a);
- Validate(ParseNetworkInfo(), NetToken_a);
- **if** *mismatch* **then**
  - Disconnect();
- **else**
  - AlertTrusted();
- **end**

end

Algorithm 4: Client resolves AP device and adjudicates

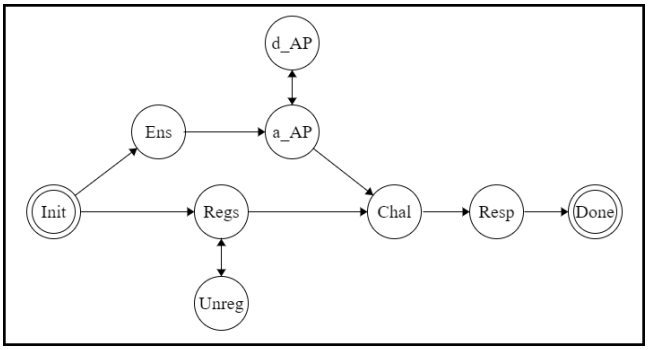

Figure 4: TrustedAP workflow FSM

(1) Chairperson deploys smart contract
(2) Chairperson enstates AP Manager
(3) AP Manager adds AP
(4) End user enters AP wireless range
(5) End user issues challenge posting digests on-chain
(6) AP receives and validates/verifies challenge
(7) AP issues response posting digests on-chain
(8) End User validates/verifies response
(9) If sound, End User stays connected. If unsound, disconnect.

4 illustrates that management AP Manager tasks 'Enstatement' and 'AP Addition' are concurrent with End User 'Registration.' It follows that this flow must be completed by all three involved parties before End Users can proceed with validation.

Supporting code for TrustedAP achieves a proof of concept for the above interaction across individual web applications for each role and supplementary scripts for client endpoint/AP off-chain interactions [6].

3 IMPLEMENTATION

TrustedAP uses several software platforms in its proof-of-concept state that facilitate its design. This section outlines the basis of the current operational model. Note that some of these platforms are subject to change on the basis of rigidity against offensive measures and achieving a future state of high-automation.

TrustedAP's smart contract code is written in the Solidity language [16]. Smart contract deployment and local deployment testing is handled using applications from the Truffle Suite [15]. Web3.js is the API that interfaces smart contract execution with the web UI through JavaScript and HTML code. Webserver mechanisms are hosted via APs and any management points (centralized or decentralized per deployment needs) using Node.js. Infura.io powers the public-facing smart contract deployment currently on the Ropsten Ethereum testnet [3, 7]. Ethereum software wallets are browser-operational through the Google Chrome MetaMask plugin [11].

The smart contract, API, and UI code for TrustedAP is objectively simple. Far more code orchestrates interactions between platforms than that which is definitive of TrustedAP (e.g., the smart contract Solidity code is fewer than 140 lines). A series of address-indexed hashmaps in-conjunction with constraints manages the entire custody chain building process. Two further 'struct-bearing' hashmaps

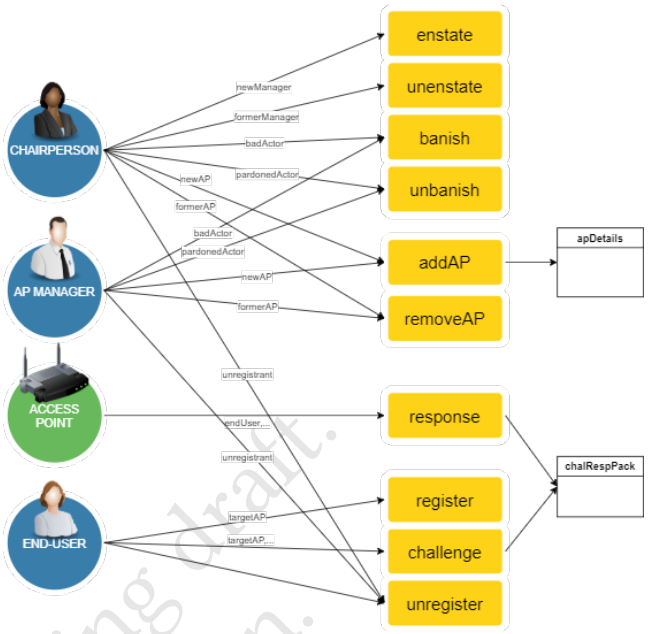

Figure 5: TrustedAP's smart contract use-case by role diagram

TrustedAP

address chairperson
struct ap_details(bytes32 apDataHash, address owner, address chair)
struct challengeResponsePack(uint challengeSeed, bytes32 hashOfResponse, bytes32 hashOfPK, address challenger, address responder, uint state)
mapping (address=>apDetails) public apIdentifier;
mapping (address=>challengeResponsePack) public getChalResp;
mapping (address=>address) enstatedBy;
mapping (address=>address) ownedBy;
mapping (address=>address) registeredTo;

modifier onlyChairperson modifier onlyAdmins modifier onlyAP modifier validEndUser modifier connectedEndUser
constructor() enstate(address newManager) onlychairperson(){} unenstate(address formerManager) onlychairperson(){} addAP(address addedAP) onlyAdmins(){} removeAP(address formerAP) onlyAdmins(){} register(address targetAP) onlyEndUsers(){} unregister(address unregistrant) onlyAdmins(){} unregister() connectedEndUser(){} banish(address badActor) onlyAdmins(){} unbanish(address pardonedActor) onlyAdmins(){} challenge(address targetAP, uint chSeed) connectedEndUser(){} response(address endUser, bytes32 encResponse, bytes32 publicKey) onlyAP(){}

Figure 6: TrustedAP smart contract diagram

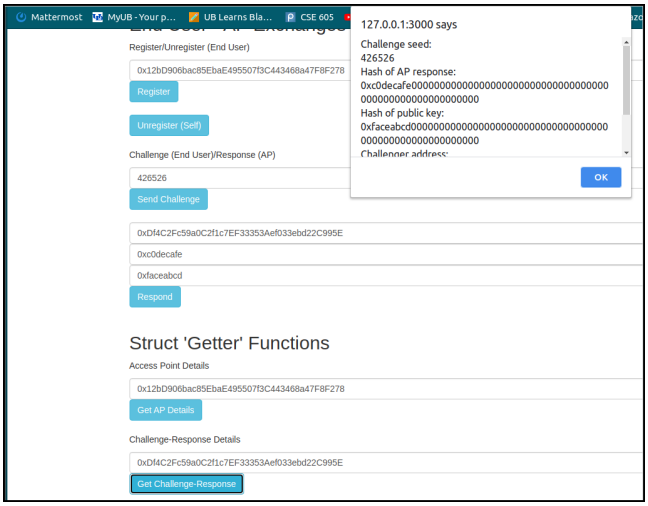

Figure 7: TrustedAP on-chain challenge-response struct

facilitate AP identification and storing challenge-response digests. The API and UI code is boilerplate, providing typed fields and buttons tied directly to smart contract function execution via the Web3.js intermediary. We show an example of the publicly deployed UI in action in Fig. 8.

The current state of off-chain message passing uses Bash scripting at this time. OpenSSL libraries manage hashing and encryption primitives. Netcat libraries (not ideal due to reverse shell vulnerabilities) facilitate message passing in plaintext between devices. Note that plaintext message passing does not undermine the security posture of TrustedAP since the verification/validation metrics are *exclusively* integrity-preserving. Thus, message eavesdropping is not controlled against although this improvement is not entirely ruled out for future development.

Each entity (Chairperson, AP Manager, AP, End User) is uniquely identified by a 160-bit address. These addresses are mapped by relationship, namely AP Managers-to-'enstater' (chairperson), AP(s)-to-installing AP Manager, and client(s)-to-AP. Further data structures exist to facilitate off-to-on-chain operations as explained in 2.1. Modifiers are used for enforcing operations may only be used by respective roles as shown in 5's Use Case Diagram. Each function in the diagram is parameterized as shown in 6.

We reinforce TrustedAP with a banish feature in which a Chairperson or AP Manager may disallow an End User from registering or thusly pairing with an AP. This feature aims to prevent malicious End Users from overwhelming APs by abusing the registration or challenge protocol. These attacks could overwhelm APs in a denial of service capacity or overspend gas needlessly. The banish feature is reversible via the function unbanish when executed by a Chairperson or AP Manager.

The primary TrustedAP smart contract struct is shared by APs and End User client endpoints which houses the challenge-response posts. Posts to the struct such as hash digests of off-chain messages require gas payments (and are potentially End User payable as mentioned in section 4.2.3). Conveniently, get-requests to this particular struct are instant and require no gas. We show UI access in 7

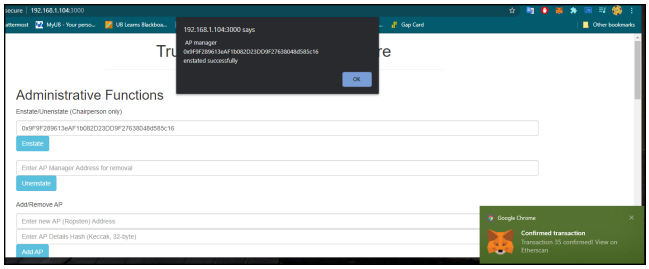

Figure 8: TrustedAP UI and operation

We note that some operations can be facilitated through the smart contract itself such as Keccak hashing and verification. However, we find that maximizing off-chain isolation helps obscure AP identification data prior to public posting, reduces gas (processing costs) and reduces the moving parts necessary for on/off-chain sync, often speeding up the challenge response process. In short, TrustedAP requires a non-trivial off-chain workload already, and adding to such scales well, affordably.

4 EVALUATION

TrustedAP's evaluation is fivefold on the following bases in which we consider evaluation criteria "layers" outlined as follows:

(1) Functionality: Is TrustedAP publicly deployable?
(2) Mathematics: Are TrustedAP interior functions cryptographically sufficient?
(3) Rigidity: Can attackers undermine TrustedAP's functionality?
(4) Performance: Is the TrustedAP false positive/negative rate acceptable and corresponds to an effective window of runtime?
(5) Transparency: Is TrustedAP affordable and hands-free enough to encourage wide adoption?

TrustedAP so far shows success in some layers and requires further development in others. This section outlines TrustedAP's current state.

4.1 Results

4.1.1 Public Deployment. The preliminary test establishing functionality for TrustedAP is its migration to a public deployment via Infura on the Ropsten testnet as described prior. The public deployment mirrors local testing as functional and bug-free. Public functionality was made available via an Infura.io hosted Ropsten testnet at Ropsten contract address:

`0x9FC1E85FAd9034597B37BcA4eD2dab37849eb161`[14].

Truffle is used to migrate the smart contact to Infura much like local testing deployments. The truffle code for this public deployment is not accessible as it would leak the author's (smart contract deployer's) wallet information. Success in this deployment (partially shown in 8) is evidenced in the cited transaction history on this address. We find that results yield a successful proof-of-concept for TrustedAP; at a minimum, End Users can hand-populate the web UI with values established from off-chain protocols. Of course,

this approach requires far too much manual entry and is not the intended endstate for TrustedAP.

4.1.2 Cryptographic primitives. On-chain, TrustedAP relies on ECDSA encryption primitives via Ethereum blockchain verifications and account creation via MetaMask. We assume that these are sound implementations of mechanisms described in the original Johnson et al. 2001 white paper showing an infinitesimal chance of reasonable-time key cracking or, by extension, address collision [8]. There is little reason to believe that these mechanisms are faulty as the entire multi-hundred-billion dollar asset-capped Ethereum blockchain requires their soundness to persist.

Off-chain, TrustedAP makes exclusive use of OpenSSL libraries for RSA private-key signing and public-key verification alongside the SHA3/Keccak hashing algorithm for posts and verifications against on-chain data [4, 5]. Again both algorithms are well-known as cryptographically strong and collision resistant [1, 13]. we assume these libraries are sound on the basis of their wide implementation across the programming spectrum for decades. Since above exhaustively accounts for the basis of TrustedAP's integrity preserving operation, we find that TrustedAP is cryptographically sufficient for its aims.

4.2 Future Work

4.2.1 Resistance to attack. We find that TrustedAP's resistance to offensive measures are dependent on its off-chain network programming libraries and the 'network perspective uniqueness' of the network tokens passed between APs and clients. As mentioned briefly prior, *open* port listening using Bash's Netcat library is susceptible to reverse shell exploits such that the current phase of development is useful for proof-of-concept alone. This issue will be remedied in future versioning with stronger libraries.

Network perspective uniqueness dictates TrustedAP's resilience to network traffic replay attacks (i.e., Man in the Middle (MitM)). For instance, a single arbitrary or randomized network token can be intercepted and replayed two-ways by a single attacker device with some difficulty. This MitM attack is successful through replying intended AP traffic to a client endpoint while maintaining a victim client endpoint connection to a rogue AP.

Strengthening the network token entails communicating details about each device's network perspective (E.g., the current version network token: `(own IP || own MAC || dest IP || dest MAC)` gathered through network discovery tools). This stronger token requires two attacker devices spoofing both a client endpoint *and* AP towards each intended/genuine device. A successful MitM, in light of this countermeasure, requires substantial attacker sophistication. Even stronger network tokens may include artifacts from self-signed certificates which are considered seriously for future development.

4.2.2 Timlieness, liveness, and safety. TrustedAP requires a 0% false positive rate (i.e., no untrusted APs marked trusted) and a very low false negative rate for success (assuming working network connections). Further, the average time in which an adjudication (the 'stay connected or disconnect' decision) is reached must be sufficiently short. This window of time corresponds to how long a client endpoint should listen for an AP response before defaulting to an 'untrusted' adjudication and disconnecting. An intuitive timing (upper-bound) threshold reflects the average length of time an attacker can perform a cursory network vulnerability scan on a target device. Network tool NMAP is capable of executing an "Intense" scan of a known remote target in fewer than 15 seconds [9]. TrustedAP lacks sufficient automation required to gather data reflecting this evaluation criteria at this time.

4.2.3 Affordability and Transparency. TrustedAP can only exceed the field of ETA mitigation solutions by overcoming the challenges posed by all 5 layers of evaluation criteria proposed here. Affordability facing public WiFi providers is very feasible since 'payable' structures can collect tokens from End Users and distribute them across AP wallets. This means End Users pay into the operational cost of TrustedAP. In fact, TrustedAP *can* operate for-profit on the mainnet if the customer appetite is sufficient. It follows that the installation time and management configuration difficulty of TrustedAP will decide whether or not it is worthwhile to public WiFi providers. The onus falls on the developer to ensure that TrustedAP is sufficiently portable (e.g., one semi-centralized management server per organization) with a hands-off installation workflow.

Thus, End User transparency is the greatest hurdle facing TrustedAP's success beyond academic proof-of-concept. We suspect that End Users will not accept spending Ethereum mainnet tokens in excess of some number of cents per connection. Further, we suspect that a perceived 'hassle' in acquiring free Ethereum testnet tokens (e.g., from testnet faucets) could undermine whether or not TrustedAP is accepted. However, the frontiers of automating the funding of End User testnet wallets are yet to be explored. We hypothesize that acceptance can be forcasted by a wide enough survey that establishes End Users' general security appetite. TrustedAP will reach this evaluation phase after showing sufficiency in the prior 4 layers of evaluation criteria.

5 DISCUSSION

5.1 Limitations

5.1.1 AP automation. TrustedAP must still be operated 'by-hand' in its proof-of-concept phase which is ultimately most troublesome for the AP operational flow. As-is, it is infeasible to suppose that an AP 'attendant' will fill in fields and click through AP responses to requesting client endpoints. The MetaMask wallet implementation serves as a boon for human Custody Managers and End Users, but is insufficient for APs. We aim for implementing a series of new libraries so AP actions are eventually hands-off (and safe).

5.1.2 A twofold lack of simple OS integration. So far, TrustedAP's AP-server software iteration may be limited to exclusive operation on a separate AP-networked device (e.g., an ethernet-connected Raspberry Pi dongle) rather than on an AP itself. This issue is mitigated by installation directly to an AP's (presumably Linux) operating system, but direct installation functionality is not always commercially available for wireless APs without substantial overrides (i.e., jail-breaking).

A client-side OS issue complicates this picture further for TrustedAP; a strictly application level TrustedAP implementation

requires some connection time between an End User and a potentially malicious AP. Ideally, clients sends a challenge to an AP *before* a complete wireless client-to-AP connection is established. A third party TrustedAP wireless connection manager, or better, major OS integration provides TrustedAP means for maximum effectiveness while reducing need for End User manual intervention post-adjudication.

5.1.3 Downtime and mainnet gas price. Ethereum testnets (The TrustedAP deployment target) are subject to semi-arbitrary downtime; the Ethereum mainnet is not. This limitation affects functionality, performance, and transparency evaluation criteria layers. We note that the deployment of TrustedAP's current state is not ideally affordable on the Ethereum *mainnet* (0.037 ETH = $88.10 at the time of this submission). The TrustedAP code structure already offloads address passing, key generation, encryption, encrypted response passing, and subsequent decryption mechanisms to off-chain operations which are relatively 'free'. The dynamic (oft increasing) value of Ethereum (and other cyrptocurrency tokens) buttresses the aim for TrustedAP's permanent deployment on Ethereum testnets (e.g., Ropsten, Kovan) rather than the mainnet. However, a semi-arbitary and unpredictable measure of testnet downtime undermines the ultimate reliability of TrustedAP as a 24/7 service. We note that smart contract deployment price is contingent on Ethereum gas price which is subject to scaling for affordability, and circumstances could yield a good case for mainnet implementation in the future.

5.2 Contribution

TrustedAP stands to pose the greatest benefit to public WiFi environments such as restaurants, hotels, and airports due to the default configuration of current generation OS-embedded WiFi managers. In short, the human factor vulnerabilities of non-public WiFi environments (home networks, businesses, schools, etc.) yield very little residual risk since connections are almost always vetted. Non-private networks are resistant to ETAs on the basis that they save connections with secret credentials. Thus, End Users will under most cases a) auto-connect to stored, intended networks with total transparency and b) client endpoint WiFi managers will not automatically connect to a rogue device unless it responds to the saved, secret credential structure (*highly* unlikely).

Public WiFi providers can send an explicit message to their clientele that they care about End User device security and data protection using TrustedAP as a medium. The costs of TrustedAP implementation are potentially constant, but the benefits scale at least linearly with the size of a positively affected customer base. TrustedAP is aiming at a fraction of residual risk posed by ETAs at the turn of the millennium, but we welcome the prospect that ETAs can finally be considered wholly obsolete in the near future.

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation under Grant No. DGE – 1754085.

REFERENCES

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. 2013. Keccak. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 313–314.

[2] Eric Y. Chen and Mitsutaka Ito. 2009. Using end-to-middle security to protect against evil twin access points. In *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks Workshops*. IEEE, https://doi.org/10.1109/WOWMOM.2009.5282395, 1–6. https://doi.org/10.1109/WOWMOM.2009.5282395

[3] Etherscan. 2021. TESTNET Ropsten (ETH) Blockchain Explorer. https://ropsten.etherscan.io/

[4] OpenSSL Software Foundation. 2018. EVP_sha3_224. https://www.openssl.org/docs/man1.1.1/man3/EVP_shake256.html

[5] OpenSSL Software Foundation. 2018. rsa. https://www.openssl.org/docs/man1.0.2/man1/openssl-rsa.html

[6] Phil Fox. 2021. *TrustedAP*. Retrieved April 17, 2021 from https://github.com/pcfox-buf/TrustedAP GitHub repository.

[7] Infura Inc. 2021. Title Page. https://infura.io

[8] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *IJIS* 1, Article 1 (2001), 36–63 pages. https://doi.org/10.1007/s102070100002

[9] Gordon Lyon. 2020. Zenmap GUI User's Guide. https://nmap.org/book/zenmap-scanning.html

[10] Alfredo Matos, Daniel Romão, and Paulo Trezentos. 2012. Secure hotspot authentication through a Near Field Communication side-channel. In *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, https://doi.org/10.1109/WiMOB.2012.6379169, 807–814. https://doi.org/10.1109/WiMOB.2012.6379169

[11] MetaMask. 2021. A crypto wallet & gateway to blockchain apps. https://metamask.io/

[12] Omar Nakhila, Erich Dondyk, Muhammad F. Amjad, and Cliff Zou. 2015. User-side Wi-Fi Evil Twin Attack detection using SSL/TCP protocols. In *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, https://doi.org/10.1109/CCNC.2015.7157983, 239–244. https://doi.org/10.1109/CCNC.2015.7157983

[13] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[14] Etherscan (Ropsten). 2021. Contract address 0x9FC1E85FAd9034597B37BcA4eD2dab37849eb161. https://ropsten.etherscan.io/address/0x9FC1E85FAd9034597B37BcA4eD2dab37849eb161

[15] ConsenSys Software. 2021. Sweet Tools for Smart Contracts. https://www.trufflesuite.com/

[16] Solidity. 2021. v.0.8.3. https://docs.soliditylang.org/en/v0.8.3/

[17] Avinash Srinivasan and Jie Wu. 2018. VOUCH-AP: privacy preserving open-access 802.11 public hotspot AP authentication mechanism with co-located evil-twins. *Int. J. Secur. Networks* 13 (2018), 153–168.