eWitness: An Architecture for Forensically Sound Private Evidence Submission[*]

Kevin Gallagher
New York University
2 Metrotech Center
Brooklyn, New York University 11201
kevin.gallagher@nyu.edu

Shweta Jain
CUNY's John Jay College of Criminal Justice
524 West 59th. Street
New York, New York 10019
sjain@jjay.cuny.edu

Brendan Dolan-Gavitt
New York University
2 Metrotech Center
Brooklyn, New York 11201
brendandg@nyu.edu

Nasir Memon
New York University
2 Metrotech Center
Brooklyn, New York 11201
memon@nyu.edu

ABSTRACT

We present eWitness, an architecture that turns a smart phone or other such mobile device into an authenticated camera for forensically sound evidence acquisition and submission. In addition, eWitness provides privacy for witnesses, allowing them to submit evidence under the protection of anonymity and thus without fear of retribution. We provide a threat model and introduce a design that preserves witness privacy while protecting the integrity of the evidence file and it's spatio-temporal context against each threat. The purpose is to ensure that the evidence self certifies its authenticity and hence is admissible in a court. In eWitness, the integrity of the evidence file is proved using robust hashing, the temporal context is preserved using a time stamping service implemented as a block chain, and the spatial context is certified through a location attestation service. These three components together strengthen confidence on the forensic soundness of the evidence and neither reveals the witness identity or location. In this paper, we present the design details of eWitness components and evaluate the performance of the time-stamping blockchain service that uses a hashgraph based consensus process. We find that this blockchain is efficient and reasonable for practical use.

KEYWORDS

privacy, blockchain, forensics, mobile computing

ACM Reference format:
Kevin Gallagher, Shweta Jain, Brendan Dolan-Gavitt, and Nasir Memon. 2020. eWitness: An Architecture for Forensically Sound Private Evidence Submission. In *Proceedings of T, 2017, T ( ),* 8 pages.
DOI: 10.475/123_4

[*]Produces the permission block, and copyright information

1 INTRODUCTION

The introduction of mobile computers has revolutionized one's participation in society, opening up multiple avenues for participation in entertainment, social activities, commerce, politics, and other activities. This increased possibility for interaction in society also increases one's ability to capture media of important events, including crimes and other undesirable activity, benefiting police, journalists, and other investigators. Such digital evidence, voluntarily submitted by a bystander can be valuable to prosecute the perpetrator of a crime. Unfortunately, the rise of technology also makes it easier for one to create false media which represent events that did not occur, or alter the time-line of events that did occur in some meaningful way. Even if the content of the evidence itself is not modified, meta-data of the content, such as the location a photo was taken, or the time of capture, can be falsified. Since tampering with digital files and meta-data is all too easy, an argument can be made to de-legitimize any such volunteered digital evidence that does not self-authenticate itself or is not attested by an alibi. For example, a video claiming to be shot in Congo, exposed atrocities against unarmed civilians by the Congolese military was recently in circulation in western newspapers [14]. However, the Congolese government was quick to dismiss the video by saying that it was shot in some other African country.

One solution to this problem is forensic analysis of the digital evidence file that is voluntarily submitted. In this method, forensic investigators analyze the evidence file to determine if any modifications have been made. A second commonly used solution in practice is to ask the witness to submit the entire device to an investigator. The investigator then uses digital forensics software such as EnCase [31] to download the hex image of the device storage. This allows a comprehensive look into logs or other artifacts that could indicate if the information related to the evidence files was tampered with. Both solutions suffer from several drawbacks. First, they is slow and financially expensive; they require forensic analysts to be hired, to analyze the computer files related to the evidence, potentially slowing down an investigation. In addition, these digital forensic processes are often subject to human and software error [2]. If the matter was to then go to court, the analyst would then need to be retained for expert witness duty, which adds to the cost of the investigation. For these reasons, the new rule in digital investigation requires that digital evidence extracted using forensic software should be self-authenticating in order to eliminate the need to call the expert witness [23]. In addition, the second solution eliminates privacy for the witness since the investigator now has access to all personal data that belongs to the witness. This potentially makes them vulnerable of being subject to ridicule in the court and retaliation from the offender.

In this paper, we present eWitness, an architecture that turns a smart phone or other such device into an authenticated camera [18], capable of capturing digital multimedia evidence which is self-certifying and tamper proof. The purpose of eWitness is to provide reasonable assurance of the integrity of the file and attesting the temporal and location contexts associated with the event in question. This assurance is necessary to prove the forensic soundness of the evidence in a court.

The rest of the paper is organized as follows. We present the threat model in Section 2, related work in Section 4 and the eWitness architecture that is able to counter these threats in Section 3. The functional specification and performance evaluation are presented in Sections 5 and 6 respectively. We finally present the limitations (Section 7), future work (Section 8) and conclusions (Section 9).

2 THREAT MODEL

Despite assuming that the witness is willing to submit the evidence, we assume that neither the witness of a crime nor the investigator of a crime can be fully trusted. Witnesses have the potential to alter evidence before submitting it to investigators to provide false assurances of innocence, place the blame for a crime elsewhere, or provide false clues to mislead the investigators. Similarly, an investigator may alter evidence for similar reasons, or to provide a stronger case for the prosecutor. If such adversaries are present, a genuine, untampered evidence file is subject to suspicion and scrutiny by the attorneys in the court. To address these issues, the validity of the content must be maintained by a third party. In the world of crime, however, there are many powerful actors. If a single third party verifies the content of submitted evidence, it may be susceptible to blackmail, bribery or political retributions.

eWitness is implemented as a client-server architecture where the clients are witnesses that submit evidence and investigators who analyze the evidence. Servers are maintained by a consortium of non-profit organizations and activist groups. We assume that both clients and a subset of the consortium servers can be compromised by an adversary or can turn malicious due to their own conflict of interests.

2.1 Compromised or Malicious Server

In our model, a compromised server has one or more of the following goals:

- `Modify entries:` This would allow the adversary to force the compromised server to destroy the reputation of a submitted piece of evidence, or to attempt to verify their own potentially doctored evidence instead. This is done simply by editing the entries on the server's storage or in memory.
- `Delete entries:` This would allow the adversary to force the compromised server to revise the time-line, also discrediting a piece of evidence or silencing it all together. This can be done by selectively deleting entries on the server's storage or in memory.
- `Deny access to users:` This would allow the adversary to compromise the server so that it is biased in accepting evidence or limit the types of events the clients can submit evidence about. This also allows for denying evidence of an event all together. This can be done by IP blocking or by taking servers offline.

Therefore, the compromised servers that we consider become actively malicious.

2.2 Compromised or Malicious Client

In our design, we consider two types of clients: the witness and the investigator. The witness captures evidence of an event, provides the eWitness service with proof of the existence of this evidence, and then provides the evidence to the investigator. The investigator then receives evidence of an event from the witness and uses the eWitness service to verify the authenticity of the evidence. This leads us to two different classes of attacks.

- `Witness Client:` A malicious witness client wishes to upload convincing altered evidence to the eWitness server, or to the investigator. This can be done by altering a picture taken from a crime scene before uploading to the service, or by sending altered evidence to the investigator, or both.
- `Investigator Client:` A malicious investigator client wishes to deny the existence of evidence, falsely claim that evidence has been altered, or falsely introduce altered or fabricated evidence. This can be done by claiming that the evidence was never received by the investigator, by claiming that the investigator determined that evidence had been maliciously altered, or by an investigator uploading a proof of existence of false evidence to the eWitness network.

3 EWITNESS DESIGN GOALS AND ARCHITECTURE

We created eWitness to provide witness with a way to privately submit evidence and investigators with a way to receive and verify privately submitted evidence. Though we assume that the witness is volunteering the information, we still recognize the importance of defending against a witness submitting false evidence. Likewise, we recognize the importance of defending against a malicious investigator who claims that evidence was tampered with, or claims that evidence doesn't exist, or who actively seeks to harm or silence a witness. Therefore, we designed eWitness with the goals of privacy and forensic soundness in mind. We achieve forensic soundness in eWitness through verifiability, reproducibility, and transparency. We achieve privacy in eWitness through witness anonymity.

There are three main components that make up the eWitness system:

(a) a smart device application with encrypted private storage space to capture and store digital multi-media evidence. The application also stores logs such as use of camera, microphone and other device sensors at the time the application is in use. Part of the verifiability and transparency goals are achieved through this application

(b) a collusion resistant time-stamping service that attests the temporal context of the evidence. This service is designed to provide transparency and supports anonymity
(c) a location attestation service that validates the location context of the evidence recorded by on-board sensors on the device. This service should support anonymity, verifiability and reproducibility.

3.1 Content Verifiability

For a piece of evidence to be accepted as reliable, it must be reasonably verifiable. In other words, the receiver of the content must be able to determine if the content he or she received has not been altered or modified in any way, within a certain margin of error. Verifiability is necessary for multiple reasons. Firstly, verifiability allows investigators and interested parties to avoid wasting their time on leads derived from false evidence. Secondly, verifiable evidence is more easily accepted in courts. Thirdly, verifiability does not allow a malicious investigator to discredit a valid piece of evidence, or to introduce fake evidence. Lastly, verifiability avoids the need for determining witness credibility. This last benefit is very important, since eWitness allows for anonymous submission as discussed later in this section. The content verifiability is supported by the eWitness application through the use of robust hashing to ensure file integrity and secure maintenance of device logs. At the time of installation, the application generates a private-public key pair. The application user, depending on her privacy needs, registers with the eWitness system to create a personal or public profile. A public key is archived in the profile and the private key is stored in the application. The eWitness application users capture digital evidence in the form of pictures, videos and voice recordings using the application. Each evidence is stored in the application's internal storage and the file is hashed using a robust hashing scheme[33]. The hash is signed using the unique private key that is generated during installation. The signed hash of the evidence files can be later used as a proof that the file is the un-tampered true original. The application then obtains a location proof from the eWitness location attestation service to tie the evidence to the location of the event.

3.2 Transparency

Transparency is an important part of any forensic investigation. It allows investigators on all sides of an incident to determine how the evidence was verified by an investigator, allowing each investigator to construct a chain of evidence. Without transparency, systems can be abused to benefit the ones in charge of the investigation, making the investigation unreliable at best and malicious at worst. Therefore, eWitness requires all operations to be transparent to all people wishing to audit the system. This allows for accountability of all operators, and can lead to an increase of trust in the eWitness system. In addition, transparency allows for the quick identification of attacks and flaws in the system. Lastly, transparency increases verifiability, since people reviewing evidence can be sure that the servers acted in good faith.

The eWitness application, uploads the evidence digest, containing the robust hash of the file and that of associated information obtained from the device sensors and the location proof, to a set of miners in the eWitness evidence mining network. Miners are distributed servers maintained by a consortium of non-profit organizations and possibly by general public. Miners execute a consensus protocol to ensure that all digests that were received are archived in the blockchain, digests are not modified, previously included digests are not moved and none of the legitimate eWitness users are maliciously denied service by colluding miners. A block is built containing the digest, the time at which the digest was first received as well as the proof that it was received by and voted for by atleast $\frac{2}{3}^{rd}$ of the miners. This block is added to the publicly available eWitness blockchain. The time stamp t of the block, obtained from this blockchain, attests that the evidence was captured at some time $t\prime$ which is no later than the time t.

In this paper, we focus on the eWitness time-stamping service.

3.3 Anonymity

Lastly, eWitness offers privacy to individuals submitting evidence to investigators or interested parties. As mentioned above eWitness provides privacy through anonymous submission. Anonymous tips are a well established legal tradition in the United States, and eWitness aims to continue this tradition and expand it to anonymous submission of evidence. This is important to protect witnesses from potentially dangerous individuals that are the subject of investigations. In addition, anonymous evidence submissions allows one to provide evidence of corruption to interested parties without fear of retribution. Thus the application only uploads a hash to the miners rather than the entire evidence. This hash does not contain any meta-data or identifying information, other than being signed using the private key. The public key is archived during the eWitness registration process where anonymous registration is also supported. An eWitness device can obtain its location through passive scanning and then obtain location attestation anonymously using crowd-sourcing or network measurements. This component of eWitness is left for future work. In the related work section 4, we have listed several prior work on anonymous location attestation.

4 PREVIOUS WORK

The only system that comes close to eWitness is the SecureDrop [1], an open source software developed by the late Aaron Swartz, an entrepreneur and Internet hactivist. The software is now maintained by the Freedom of Press foundation. The SecureDrop is individually used by different newspapers to allow anonymous communication with whistleblowers and under-cover journalists. This service is safe for the users as it allows high degree of privacy, however there is no way to ensure the truthfulness of the data being submitted. Therefore, the main difference between SecureDrop and eWitness is that an evidence submitted through the latter has features that make them self-certifiable.

In this section we present related work in three areas: techniques to prove file integrity, time-stamping services that are proposed for crypto-currencies and techniques that generate location proofs using crowd-sourcing and network measurements. These form the three components of the eWitness that make the evidence self-certifiable to a large extent.

4.1 File integrity

In normal applications, file integrity has typically been guaranteed by secure hash funcitons, or one way functions that take arbitrary length binary input and map it to a fixed length output. In order to be a cryptographically secure hash, the function must have the following properties: [24]

(1) preimage resistance
(2) 2nd-preimage resistance
(3) collision resistance

Though these properties work very well for the integrity of arbitrary file types, they tend to be limited in usefulness for images. Many images undergo many different operations that do not significantly alter the content before and after they traverse the Internet. These operations include geometric transformations, compression, de-noising, format conversion, and others. Though these operations do not meaningfully alter the content of the file, they do produce significantly different hashes when traditional hashing methods are used. To authenticate images, one must use a hashing algorithm, called a robust image hashing algorithm, that is immune to such transformations while being sensitive to operations that affect the content of the image [29]. Many different image hashing algorithms exist, using techniques such as Fourier transformation [26], Zernike moments [33], and more. For the purposes of generality and forward-compatability, we do not make any assumptions about the specific robust image hashing algorithm used, except that it defends against meaningful alterations of the content such as cropping, additions, deletions, and changes to already existing picture content.

4.2 Blockchains for distributed time-stamping

Crypto-currencies have given a new life to research in distributed, asynchronous consensus. The most prominent consensus system used today is the bitcoin miner networks which is based on proof-of-work [4] which requires the user to solve a cryptographic puzzle. The user computes a string that when hashed generates a value with a certain number of leading zeros. The problem of computing the string is computationally extensive but verification is cheap. In Bitcoin blockchain, the proof-of-work is used to implement a distributed timestamp server that maintains a public ledger of the time order of transactions that used the crypto-currencies. Each transaction is sent as a broadcast to the entire mining network. Miners compute the next block using the transactions they receive. The first miner to compute a block, broadcasts it to the others. If the block can be verified, other miners convey their acceptance by using the hash of this block to work on the next block. This timestamp ledger acts as a safeguard against multiple spending of the same coin. Proof-of-work requires heavy compute resources and therefore reversing a block requires re-doing the work. If the block chain has advanced, any alteration or tampering may need computation of several blocks. Therefore, the integrity of the blockchain is guaranteed by the computational difficulty. The system requires a minimum of 51% miners to turn rogue in order to successfully compromise the integrity of the system. Therefore, the integrity is largely dependent on the availability of large number of miners which reduces the possibility that a cartel could take over by acquiring 51% of the mining power of the network. However, as the minting rate of crypto currency reduces, the transaction fee needs to be increased accordingly in order to maintain the incentives that attract a large number of miners needed to ensure security. If the transaction fee falls close to zero, the system may suffer from the tragedy of the commons with honest miners leaving the mining network, making it vulnerable to the 51% attack.

An alternative distributed consensus is called proof-of-stake [7] where the system consists of several currency holders who lock their currency for a certain amount of time, i.e., place a stake. In order to extend the consensus, the stakeholder sign the next block/extension. The proof-of-stake algorithm may randomly selects some currency holders to sign an extension. The system is secure as long as the stakeholder is locked in for a sufficient amount of time that prevents a stakeholder from first cashing out his stake and then create a new fork starting at the point in history where the stakeholder had control. The security of the system still depends on having a critical mass of miners to reduce the possibility of collusion. A hybrid method called Ppcoin was proposed [19] in 2011 where the system consisted of two blocks. The first type of block, called kernel, is generated using the proof-of-work method. The miner who generates the kernel then creates a special transaction called coinstake in which the miner pays a stake to gain the privilege of generating the next block. The generation of the kernel using proof-to-work introduces a stochastic process that ensures random selection of stakeholder. Several other systems based on proof of stake have also been proposed [8, 28].

Byzantine fault tolerance (BFT) has been studied in the context of distributed processing and distributed databases [10]. A BFT based system functions correctly as long as the number of faulty entities is less than one-third of the total. When the conditions of fault tolerance is achieved, the system is said to have quorum. In the context of public ledgers such as in our application and in crypto-currencies, a Federated Byzantine agreement system (FBAS) has been proposed [21, 25] in order to reduce the latency involved in achieving quorum with a large number of miners. Nodes participating in FBAS select their own trust group to form quorum slices. A single node may be a part of one or more such slices. In order to have protection against collusion attacks, the network must have quorum intersection i.e., there may not be any pair of quorum slices q_i, q_j with membership sets v_i, v_j such that $v_i \cap v_j = \emptyset$. This requirement ensures that no two quorum slices accept contradictory views of a block. In addition, when byzantine nodes are present, a robust FBAS system with a set of nodes V will consist of a dispensable set $D \subset V$ such that the quorum slices formed from the set V
D has quorum intersection as well as quorum. D may contain the set of byzantine nodes B as well as nodes that have been blocked by nodes in set B from acting correctly.

A hybrid proof-of-stake and BA system was proposed to enable consensus without mining (Tendermint) [6, 20]. Thus, instead of sending transactions as broadcasts to the network, they can be sent to a single node in the network. Transactions are validated by users who have coins at stake, deposited as a bond deposit using a bond transaction. Thus validators have the incentive to only sign valid transaction because if found guilty of signing a fraudulent transaction, they might be forced to forfeit their bond. A block consists of valid transactions and signatures. Only blocks with a

2/3 majority of validator signatures is considered committed to the ledger. The signatures are obtained through a partially synchronous BA algorithm that assumes an upper bound on message latency. The consensus system ensures that if a fork is created, both branches may not consist of all valid transactions i.e., one of the branch must contain some duplicitous validator signatures. Any block holder can detect the guilty validator. As long as the guilty is found within the unbonding period i.e„ before the validator cashes his bond out, the system can recover from the fraudulent transaction. In order to thwart long range double spend attacks, users must re-sync their blockchain continually within the unbonding period.

More recently a hashgraph based byzantine agreement protocol was proposed by Baird [5]. In this scheme, a user sends a hash to one of the miners in the network. The receipt of the hash is denoted as an event. The miner as well as the user re-broadcast the event to additional miners i.e., gossip about the event to k other miners. This leads to the gossip of the event propagating at exponential rate and thus the network reaches consensus very quickly. Each gossip has a chain of signatures from previous gossipers, and therefore a chain with two-thirds of the miners' signatures is sufficient to declare consensus. Thus a final vote is not needed reducing the overhead significantly.

4.3 Location Proofs

It is important to tie a digital evidence with the location of the event that is being presented in the evidence. While most smart-devices are capable of obtaining their location through various publicly available localization infrastructure, unfortunately it is also all too simple to spoof the location by synthetically generating satellite, wireless LAN and cellular signals [27, 30]. Davis et al [11] suggest using a single third party, similar to a certificate authority, as an alibi and using cryptographic means to protect privacy. However, this solution suffers from replay attacks, collusion to falsify the location and to compromise the location privacy of the user. Arunkumar et al [3] suggest a solution where all devices passively scan and report each others presence along with their locations. If used globally and in a large scale, this solution can first establish trust about devices that are often truthful and then use the alibi provided by the trusted devices. Similarly, Gambs et al [13] present a solution where location proofs are obtained from a group of witnesses who are present around the user. However, both solutions require a critical mass of participants in all locations. In addition, these solutions suffer from collusion attacks who could report false locations or compromise the location privacy of the user. However, an alibi based crowd-sourcing infrastructure for location proof can gather a critical mass if several real-time location based services such as ride hail, parking reservations, delivery services etc. start suffering from location forgery. Therefore, one direction of research is indeed in building cheap and usable crowd-sourcing based anonymous location proof systems if the threat for replay and wormhole attack can be countered.

Geolocation in the network has been an area of research for more than a decade. Several solutions have been suggested that use round trip time and constraint satisfaction algorithms [15, 17] to locate Internet hosts. Other techniques such as IP geolocation [17] and proximity detection by measuring the timing of fast challenge and response [9] can be used in parallel to improve accuracy. Wong et al [32] suggest using round trip times (RTT) of data transmission from fixed anchors at known locations in the Internet to estimate distances from the device. Prior knowledge of RTT between the anchors is used to correct for queuing delays. Additional information such as IP geolocation, piecewise localization of intermediate routers and elimination of implausible locations/regions are used to create a set of positive and negative constrains. Then a circular ring is computed around each anchor. The solution space consists of a set of points bounded by Bezier curves calculated from these rings. Experiments conducted by Wong et al [32] bounds location errors to 22 miles and a follow up by Niang et al [22] shows that selecting anchors that are geographically closer to the target provides more accurate location estimates. In our solution, we face further difficulties such as use of Tor [12] by the target to conceal her location in the network. The inter-router delays in Tor can be variable and are in addition to queuing delays. However, Hopper et al [16] show that the measured latency of Tor circuits and correlation with the latency observed from the target may be sufficient to extract some information about the network host to which the target is connected. Such measurements from 2 or more hosts is sufficient to locate the target.

A network measurement technique can counter the threat of location forgery simply because it is hard for a target to synthetically construct a set of network measurements while ensuring that the forged location will fall within the error margin of the locations estimated through the network measurements. This technique preserves the location privacy of the user since the anchors themselves only compute the possible circular region, and hence are unable to pinpoint the location of the target unless they collude. Collusion can be prevented if the system and target together select a random set of anchors from a significantly large pool.

5 EWITNESS TIME-STAMPING BLOCKCHAIN DESIGN

We created eWitness to address the problem of private evidence submission with forensic soundness. Our system consists of a set S of servers and a set C of clients, which is broken into two subsets, a subset of investigators C^i and a subset of witnesses C^w. These sets are not necessarily disjoint ($C^w \cap C^i$ is not always $\emptyset$). Servers can interact with each other through any Byzantine Fault Tolerant decentralized consensus algorithms. For the proof of concept implementation, we implemented eWitness with the Hashgraph Consensus Algorithm. These participants interact through the following functions:

5.1 Submit Hashes

When a witness $c^w \in C^w$ collects evidence from a crime scene, they must hash the evidence, perform a location attestation, and sign the resulting values. The witness then can submit the created evidence to the network. First c^w chooses n servers $X = \{s_1, ..., s_n\}, X \subset S$ either at random or based on trust. When a witness captures evidence e (such as a photo) of an event of note, such as a crime or important political event, the evidence is hashed on the witness' device, giving us $h = H(e)$. Each hash will be signed by the client, generating signature $s = S(K^{pub}_{c^w}, h)$, making it impossible for the server to alter

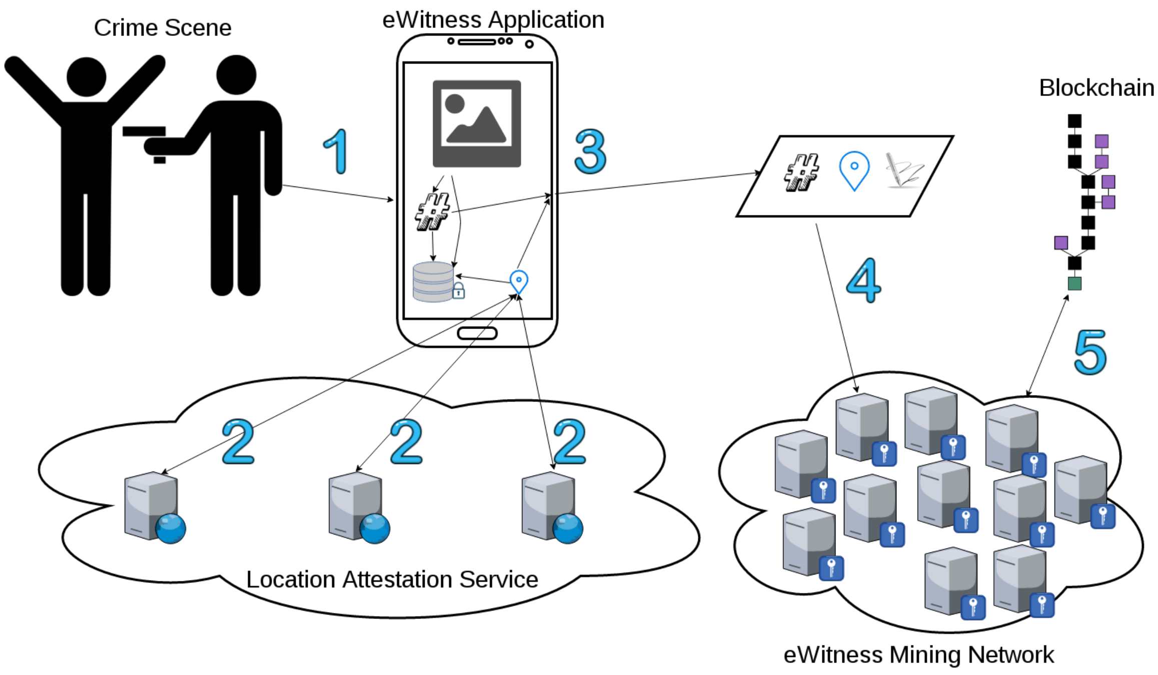

Figure 1: A witness capturing evidence of a crime and receiving a location attestation from the attestation servers. This is then uploaded to the eWitness miner network, which adds it to the blockchain.

the content of the submission without detection. Privacy preserving spatiotemporal attestation is then performed (see Subsection 4.3), giving us a location attestation l. After this is done, the client submits (h, s, l) to a server $x_i \in X$ from the set of chosen servers X, starting with $i = 1$. The client c^w then queries the status of his or her submission from a randomly selected server $v \in S$ where $v \neq x_i$ to act as a verifier. The witness does this multiple times to act as a verifier. If the witness sees his or her content has successfully propagated, the witness halts successfully. If the witness does not see his or her content in the network within a configurable time period or a configurable number of failed verification attempts t, the witness selects the next server and attempts to submit the content again. This process protects against benign events such as server failures as well as denial of service by malicious servers.

Using the hash h of the evidence content e makes the system tamper evident. If the witness attempts to give doctored evidence e' to an investigator, the investigator will hash the altered evidence e' and get h'. The investigator will be able to determine that the evidence is not genuine by observing that $h \neq h'$. Likewise, if a server x_i attempts to modify the file in transit it will be detected based on the same procedure above. Lastly, denial of service to the witness or the investigator is solved through having n servers $\{x_1, ..., x_i, ..., x_n\} \in X$ responsible for the content. The consensus mechanism of these servers will be discussed in more detail below. Figure 1[1] demonstrates the process of evidence capture, hash creation, and location attestation.

[1]The crime scene image was created by author Abu Badali. The blockchain image was created by Theymos from Bitcoin wiki vectorization.

5.2 Add Content

When a server receives a content submission (h, s, l), it generates a proxy URL u that an investigator can use to fetch the content that the witness holds, which is returned to the witness as part of a success message. It then adds the hash h, the signature of the hash s, and spatiotemporal commitment l, and the proxy URL u to a list of hash entries H. The server then enters the consensus stage.

5.3 Hash Consensus

Our service is capable of using any Byzantine Fault Tolerant decentralized consensus algorithm. In our proof of concept implementation of eWitness, we used the Hashgraph Consensus Algorithm. The Hashgraph algorithm is distributed with no leader, does not require proof-of-work, is quick, and is efficient in terms of network communication. In the Hashgraph algorithm, each consensus server spreads information through gossip with other servers. Through gossip about gossip, every server $s \in S$ learns about every submitted piece of evidence with probability $p = 1$, provided that the number of malicious servers does not exceed $\frac{|S|}{3}$.

In the Hashgraph consensus algorithm, each member of the consensus group randomly chooses another member of the consensus group an sends them everything they know. The receiver then creates an event to acknowledge that they received the gossip and returns it to the sender. Each event therefore has two ancestors, a self-ancestor, which is the last event created by the receiver of a gossip, and the other-parent, which is the previous event of its communication partner. After each gossip, the nodes call three functions, divideRounds, which assigns a round number to each event, decideFame, which determines whether an event can be "seen" by a certain number of events, and findOrder, which assigns a consensus time-stamp to each event and decides on the order of events. For more details on the Hashgraph consensus algorithm, see the whitepaper [5].

6 EXPERIMENTAL SETUP, SOFTWARE DESIGN AND EVALUATION

We evaluated the efficiency of eWitness's client registration and hash creation through the use of a prototype eWitness server and a simulated eWitness client. The eWitness server ran on an Ubuntu 16.04.2 LTS Digital Ocean droplet with 1 GB of RAM, a single CPU, and 30 GB of hard drive space. This digital ocean droplet was chosen because of its cheap price of $10 per month, which is affordable by most individuals or companies who may be interested in participating in the eWitness network, though real prices would be slightly more expensive for expanded storage. The simulated eWitness client ran on a Fedora-23 Virtual Machine with 2 GB of RAM and 4 virtual CPUs. The resources on this virtual machine are less than those of a Samsung Galaxy S7 edge. Both the server and the simulated client were implemented in Python version 3.4.3.

6.1 Registration Experiments

To test the feasibility of client registration in eWitness, we ran three experiments to test the amount of time it took to register a client. For the client, the process of client generation required generation of an 4096 bit RSA keypair, creation of a cryptographic signature, and sending a registration message to the server of choice. For the server, this process involved checking that the cryptographic signature was valid. We ran this experiment client-side with 10, 100, and 1,000 client registrations, with average client registration times of 9.64 seconds, 8.55 seconds, and 8.59 seconds respectively, and an overall average of 8.59 seconds per client registration, including network latency. Though this is a high number, it is a one-time registration cost that is dominated mostly be cryptographic key generation.

6.2 Hash Experiments

To test the feasibility of client hash submission in eWitness, we ran three experiments to test the amount of time it took to submit a hash to the chosen eWitness server. For the client, the process of hash submission required calculating the SHA256 hash of a file, signing the hash with the client's RSA keypair, and sending it over the network to the eWitness server of choice. For the server, this process involved checking the validity of the cryptographic signature on the hash using keys stored from the registration experiment and storing the hash if the signature check passed. We ran this experiment client side with 100, 1,000, and 10,000 hash submissions, with average hash submission times of 0.15 seconds, 0.16 seconds, and 0.16 per hash respectively, and an average overall hash submission time of 0.16 seconds per hash, including network latency. This number is low due to the efficiency of modern cryptographic hash functions, and shows how efficient eWitness is for client devices. We leave the testing of robust image hashes to future work.

6.3 Consensus Experiments

To test the feasibility of the consensus algorithm, we ran 4 experiments to test the amount of time it took for an event to be accepted by 2/3rds of the eWitness servers, with server groups of 5, 10, 20, and 50 respectively. In these experiments, the server groups were run on the same machine and communicated via connections on localhost, and sharing a source of randomness for key generation. In addition to the consensus algorithm, each server had a one-time setup cost which included peer discovery and cryptographic key generation. The average of our one-time setup costs in our 4 experiments was 189.07 seconds. Though these setup costs are high, they are dominated by key generation for each server and will be improved drastically when run in parallel on separate machines, and will be expected to run on the order of 10 seconds. In the first experiment the 5 server eWitness network agreed on the status of a submitted hash in 52.67 seconds. In the seconds experiment, the 10 server eWitness network agreed on the status of a submitted hash in 135.95 secondst. In the third experiment, the 20 server eWitness network agreed on the status of the submitted hash in 180.05 seconds. In the final experiment, the 50 server eWitness network agreed on the status of the submitted hash in 350.32 seconds. Since these servers were run on the same physical machine and communicated over TCP connections on localhost, these numbers do not include the latency delays that would be present in a typical Internet connection.

7 LIMITATIONS

Though eWitness is a great first step towards forensically sound private evidence submission, there are still a few limitations of our work. Firstly, eWitness' reliance on using only distributed consensus on submissions of meta-data to determine forensic soundness creates no guarantees that a piece of evidence was not doctored before it was uploaded to the service. This limitation is mitigated, however, by the use of other forensic techniques that examine the content of the evidence if it has been submitted to the network a significant amount of time after the event it purports to represent has taken place.

Secondly, the location attestation aspect of eWitness could use some improvement both in efficacy and privacy. Clients using privacy enhancing tools such as Tor have the potential to frustrate our location attestation techniques, since Tor introduces latency based on the circuit chosen by the client. Similarly, a malicious client that deliberately delays RTT can make its measurements useless. Additionally, the location attestation servers are highly trusted, and malicious location attestation servers can keep logs of IP addresses and location attestation requests compromise the anonymity of users.

Lastly, our approach assumes uniform trust across all of the nodes for submission of content. Though this cannot affect the submission of the hashes themselves due to the consensus mechanism and digital signatures, it can expose a client's IP address to a malicious server run by an investigator or other interested party.

8 FUTURE WORK

Many of the limitations mentioned in Section 7 will be solved in future work.

Firstly, we intend on addressing the problem of location attestation to make it both more accurate and more private. This can be done through passive measurement of an existing infrastructure, such as cell phone networks. If a user is in a certain place at a certain time, that user has the ability to sniff and hash signaling messages on local cell phone towers. Since sniffing is passive, this action would not reveal any information that can identify the user, other than that the user belongs to a set of individuals within a certain area at a certain time. This approach, however, assumes the cooperation of third party companies or the creation of an extensive infrastructure to passively sniff cell phone signals in all areas.

Secondly, we intend on researching useful trust schemes for selection of servers for the submission of content. These schemes would be used to mitigate the threat of a client choosing a malicious server seeking to deanonymize witnesses through IP address. These trust schemes could include server fingerprinting, identity based trust schemes (a client only trusts a set of individuals or organizations), context based trust schemes (the client trusts a set of servers based on the content he or she is submitting), or other trust schemes.

Lastly, we intend on researching possible inactivation schemes to encourage individuals to run eWitness servers. One potential way to incentives running an eWitness server is to cooperate with bit-torrent trackers to allow heavy "leechers" of bit-torrent services to gain reputation as a "seeder" for participating in the eWitness network as a server.

Another potential solution is to build a peer-to-peer software for eWitness such that the software has a low footprint in terms of CPU, memory, and network utilization. People may choose to install such an application simply as a good will or altruistic task. If this approach is taken, eWitness would run a trusted, distributed third part which will monitor and approve new eWitness servers in order to minimize the likelihood of a Sybil attack and expel malicious actors.

9 CONCLUSIONS

In this paper we presented eWitness, an architecture used to turn a device into an authenticated camera for evidence collection and private evidence submission. We described the threat model of malicious witnesses and malicious investigators and how eWitness protects against these threats using robust hashing, location attestation, and Byzantine Fault Tolerant consensus. Lastly we created a prototype implementation of eWitness and demonstrated its practicality through three experiments: client registration, hash experiments, and consensus experiments. Lastly, we introduced future work that will improve the eWitness architecture.

REFERENCES

[1] Kevin Paulsen Aaron Swartz. SecureDrop: Freedom Of Press Foundation. https://github.com/freedomofpress/securedrop. (????).

[2] Lizette Alvarez. 2011. Software designer reports error in Anthony trial. *New York Times* (2011), A14.

[3] Saritha Arunkumar, Mudhakar Srivatsa, Murat Sensoy, and Muttukrishnan Rajarajan. 2015. Global attestation of location in mobile devices. In *Military Communications Conference, MILCOM 2015-2015 IEEE*. IEEE, 1612–1617.

[4] Adam Back. 2003. The hashcash proof-of-work function. *Draft-Hashcash-back-00, Internet-Draft Created,(Jun. 2003)* (2003).

[5] LEEMON BAIRD. 2016. HASHGRAPH CONSENSUS: FAIR, FAST, BYZANTINE FAULT TOLERANCE. (2016).

[6] Ethan Buchman. 2016. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains.* Ph.D. Dissertation.

[7] Vitalik Buterin. 2013. What proof of stake is and why it matters. *Bitcoin Magazine, August* 26 (2013).

[8] Vitalik Buterin. 2014. Slasher: A punitive proof-of-stake algorithm. *Ethereum Blog URL: https://blog. ethereum. org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm* (2014).

[9] Srdjan Čapkun, Levente Buttyán, and Jean-Pierre Hubaux. 2003. SECTOR: secure tracking of node encounters in multi-hop wireless networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*. ACM, 21–32.

[10] Miguel Castro, Barbara Liskov, and others. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[11] Benjamin Davis, Hao Chen, and Matthew Franklin. 2012. Privacy-preserving alibi systems. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ACM, 34–35.

[12] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router.* Technical Report. DTIC Document.

[13] Sébastien Gambs, Marc-Olivier Killijian, Matthieu Roy, and Moussa Traoré. 2014. PROPS: A privacy-preserving location proof system. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*. IEEE, 1–10.

[14] Jeffrey Gettleman. 2017. Look, They Are Dying': Video Appears to Show Massacre by Congolese Soldiers. *New York Times* (February 2017).

[15] Bamba Gueye, Artur Ziviani, Mark Crovella, and Serge Fdida. 2006. Constraint-based geolocation of internet hosts. *IEEE/ACM Transactions On Networking* 14, 6 (2006), 1219–1232.

[16] Nicholas Hopper, Eugene Y Vasserman, and Eric Chan-Tin. 2010. How much anonymity does network latency leak? *ACM Transactions on Information and System Security (TISSEC)* 13, 2 (2010), 13.

[17] Ethan Katz-Bassett, John P John, Arvind Krishnamurthy, David Wetherall, Thomas Anderson, and Yatin Chawathe. 2006. Towards IP geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 71–84.

[18] John Kelsey, Bruce Schneier, and Chris Hall. 1996. An Authenticated Camera.. In *acsac*. 24–30.

[19] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19 (2012).

[20] Jae Kwon. 2014. TenderMint: Consensus without Mining. *URL http://tendermint. com/docs/tendermint {_} v04. pdf* (2014).

[21] David Mazieres. 2015. The stellar consensus protocol: A federated model for internet-level consensus. *Draft, Stellar Development Foundation, 15th May, available at: https://www. stellar. org/papers/stellarconsensus-protocol. pdf (accessed 23rd May, 2015)* (2015).

[22] Ibrahima Niang, Bamba Gueye, and Bassirou Kasse. 2010. GeoHybrid: A hierarchical approach for accurate and scalable geographic localization. In *2010 ITU-T Kaleidoscope: Beyond the Internet?-Innovations for Future Networks and Services*. IEEE, 1–8.

[23] Committee on Rules of Practice and Procedure of the Judicial Conference of the United States. 2015. Rule 902(14). Evidence That Is Self-Authenticating. Certified Data Copied from an Electronic Device, Storage, Medium, or File. *FEDERAL RULES OF EVIDENCE* (August 2015).

[24] Phillip Rogaway and Thomas Shrimpton. 2004. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *International Workshop on Fast Software Encryption*. Springer, 371–388.

[25] David Schwartz, Noah Youngs, and Arthur Britto. 2014. The Ripple protocol consensus algorithm. *Ripple Labs Inc White Paper* (2014), 5.

[26] Ashwin Swaminathan, Yinian Mao, and Min Wu. 2006. Robust and secure image hashing. *IEEE Transactions on Information Forensics and security* 1, 2 (2006), 215–230.

[27] Nils Ole Tippenhauer, Kasper Bonne Rasmussen, C Popper, and Srdjan Capkun. 2008. iPhone and iPod location spoofing: Attacks on public WLAN-based positioning systems. *System Security Group, ETH Zürich Univ., Zürich, Switzerland, Tech. Rep* 599 (2008).

[28] Pavel Vasin. 2014. Blackcoin's proof-of-stake protocol v2. (2014).

[29] Ramarathnam Venkatesan, S-M Koon, Mariusz H Jakubowski, and Pierre Moulin. 2000. Robust image hashing. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, Vol. 3. IEEE, 664–666.

[30] Ting Wang and Yaling Yang. 2013. Analysis on perfect location spoofing attacks using beamforming. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2778–2786.

[31] S. Widup. 2014. . New York: McGraw-Hill Education.

[32] Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer. 2006. Geolocalization on the Internet through Constraint Satisfaction.. In *WORLDS*, Vol. 6. 1–1.

[33] Yan Zhao, Shuozhong Wang, Xinpeng Zhang, and Heng Yao. 2013. Robust hashing for image authentication using Zernike moments and local features. *IEEE Transactions on Information Forensics and Security* 8, 1 (2013), 55–63.