Gringotts: Securing Data for Digital Evidence

Catherine MS Redfield
SECOM Co., Ltd.
Intelligent Systems Laboratory
Mitaka, Tokyo 181-8528
c-redfield@secom.co.jp
cat_red@alum.mit.edu

Hiroyuki Date
SECOM Co., Ltd.
Intelligent Systems Laboratory
Mitaka, Tokyo 181-8528
h-date@secom.co.jp

***Abstract*—As digital storage and cloud processing become more common in business infrastructure and security systems, maintaining the provable integrity of accumulated institutional data that may be required as legal evidence also increases in complexity. Since data owners may have an interest in a proposed lawsuit, it is essential that any digital evidence be guaranteed against both outside attacks and internal tampering. Since the timescale required for legal disputes is unrelated to computational and mathematical advances, evidential data integrity must be maintained even after the cryptography that originally protected it becomes obsolete. In this paper we propose Gringotts, a system where data is signed on the device that generates it, transmitted from multiple sources to a server using a novel signature scheme, and stored with its signature on a database running Evidence Record Syntax, a protocol for long-term archival systems that maintains the data integrity of the signature, even over the course of changing cryptographic practices. Our proof of concept for a small surveillance camera network had a processing (throughput) overhead of 7.5%, and a storage overhead of 6.2%.**

Keywords – *Digital Evidence*, *Digital Signatures*, *Stream Data*, *Long-Term Authenticity*, *Evidence Record Syntax*

I. Introduction

Any company in the modern world collects data on a large scale: data generated by users, by employees, by automatic digital archiving. Where, how, and how long that data is stored depends on a number of factors, including legal necessities. Data collected in the usual course of business may become essential in a court case, at which point its status as electronic evidence depends on the trustworthiness and reliability of that data's transmission, processing, and storage [1]. Evidence must, of course, be available in a human-accessible form, however in this paper we are concerned with the trustworthiness of reliably stored data. In order to trust evidence from an interested party, the court must be reasonably certain that the data provided is the whole and entire record at the time and from the device it purports to be from.

Relevant data can be accumulated from a variety of sources, including sensor networks, user or employee machines, and access control devices such as metal detectors. One common factor among these varied systems is that they consist of many data sources sending their information to a central server or data center. This network model is somewhat unusual in security applications, which are generally intended for trustworthy multicast or decentralized peer-to-peer data distribution (rather than collection) systems.

One common form of digital evidence is the data generated by a company or organization's security sensors, e.g. surveillance cameras. While some security cameras are directly connected to analogue tapes or exclusively use a secure intranet, a growing percentage are IP cameras: cameras that stream their recordings to a predetermined web server or database using the Internet [2], [3]. However, since the data is being transmitted over a network that may not be trusted, guaranteeing the authenticity and integrity of the archived data is a serious concern. Security camera streams can be found by straightforward web searches [4], and once a stream is intercepted, altering or reordering the data silently is possible using a Man in the Middle Attack. An adversary could also generate false data and send it to the server as a valid stream, in which case the conflicting data from the same place and time might make it impossible to identify the correct record. Even if the data transmitted over the network were authenticated, and thus protected from allegations of tampering or falsification, legal processes can work more slowly than cryptographic ones. If the data is protected by an algorithm that is later deemed insecure by the legal or cryptographic communities (and every algorithm eventually becomes insecure), the data will become inadmissible as evidence.

The major contributions of the paper are: an efficient and secure signature scheme for data streamed from multiple sources to a single server; and the Gringotts System, which uses this scheme in conjunction with a variant of the Evidence Record Syntax (ERS) [5] protocol to indefinitely verify the existence and integrity of data streamed from multiple sources to a database over an untrusted network. When implemented on a small surveillance camera-style layout, the Gringotts System increased processing time by 7.5%, as compared to an unsigned data stream. On the same implementation, data stored in the Gringotts Database required 6.2% more storage space than the same data stored in a standard (unprotected and unverifiable) database.

This paper is organized as follows: §II provides context for the project and describes related research. §III describes the Gringotts Signature Scheme and lays out the overall system design. §IV analyzes the security of the Gringotts System components individually and collectively. §V describes our proof of concept implementation, and analyzes its performance. §VI concludes and discusses future research goals.

II. Background and Related Work

Digital signatures can be used to guarantee data integrity over untrusted networks. A digital signature, generally based on a public key cryptosystem, allows the sender to include an

DOI 10.1109/SPW.2014.11

encrypted version of the data (the signature) that the receiver can check against the received data. If a secure algorithm is used, only the sender has the ability to generate a verifiable signature for the data it sends, and thus if the receiver can verify a signature, it can be assumed that the data was unaltered during transmission. Because only the sender can create a valid signature (even the receiver cannot forge a signature), digital signatures also provide the property of non-repudiation of origin, which means that at some later date, the sender cannot deny having signed and sent the data.

Signatures schemes specifically intended for streamed data are not new – Gennaro and Rohatgi proposed a solution as early as 1997 [6]. One of the main motivating use cases for stream signature was and remains sending video data from a web server to multiple end users (a multicast network), where dropped packets present little risk and can problematically delay the network. As such, there are a number of signature schemes for streamed data that allow for dropped packets or build on the multicast structure [7]–[11]. Since our goal is a trustworthy system, easily dropped packets are of greater concern than optimally efficient network usage, although since video data is still an applicable use case for Gringotts, bandwidth usage remains a consideration. Our proposed system layout is the inverse of a multicast system (many senders and one receiver), so none of these schemes are applicable. While the original Gennaro-Rohatgi scheme fulfills the Gringotts security requirements by strictly disallowing dropped packets, micro-benchmarking showed that, in practice, the one-time signatures required by Gennaro-Rohatgi are slower than an efficient public key scheme such as the Elliptic Curve Digital Signature Algorithm (ECDSA); and that both the Gennaro-Rohatgi one-time signature scheme and a naive one ECDSA signature per packet scheme resulted in a sender-side processing time that was slower than the data generation speed, making those solutions impractical for a real system. The microbenchmarking of ECDSA is discussed in §V-B.

There have also been proposals for error correcting code-based signature schemes for streamed data [12]–[14]. However, all these solution are solely theoretical in nature, and would require error correcting or erasure codes significantly more efficient than those currently in use.

The eventual obsolescence of most cryptographic algorithms is a more recent, but very pressing, concern. Information theoretically secure cryptographic primitives may last longer, but are generally inefficient compared to the current best computationally secure scheme, and may also require novelty hardware to implement [15]. Working from a different point of view, the IETF Long-Term Archive and Notary Services Working Group [16] has proposed Evidence Record Syntax (ERS), a protocol for storing authenticated data in long-term non-repudiable manner [5]. The Gringotts system uses a variant of ERS that includes protection for data ordering [17]. Since ERS is designed to efficiently update its authentication algorithms as they become obsolete, a system using ERS can always use the fastest secure signature scheme without compromising the future security of the data.

III. Design

The Gringotts System is a set of cooperative software tools that can be used in conjunction with existing data gathering deployments. Gringotts assumes that multiple senders connect to a single server, which either serves as or connects to a database where the data is stored. Administrators, end users, and law enforcement professionals may connect to the database to view the saved data.

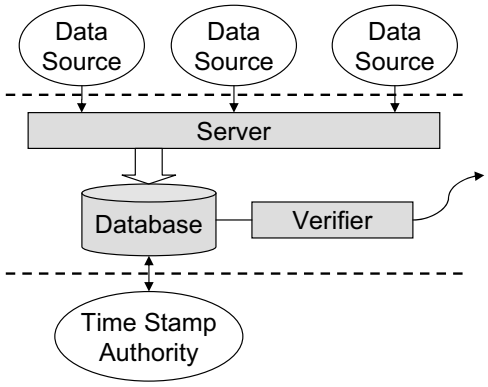

Fig. 1. Layout of a Gringotts System. Trusted data sources transmit signed information to the semi-trusted server, which stores it on a semi-trusted database. The database runs ERS, which makes use of timestamps from a trusted Time Stamp Authority. Data is accessed from the database through a verifier program.

In Gringotts, each sender has a unique key pair that it uses to sign the data it generates before sending that data over the network. The signature is verified and preserved by the Gringotts server, and archived on the database using ERS. Users with a right to access the data can connect to the database and verify the signature and the ERS metadata before reading.

A. Threat Model

Fig. 1 shows the layout of a system using Gringotts. The various data sources transmit information to the server, which stores it in the database. When data is accessed, its authentication is confirmed by the verifier.

The Gringotts threat model has three threat levels: untrusted, semi-trusted, and trusted. The network between the data sources and the server is considered to be completely *untrusted*. It is assumed that an attacker can gain access to and manipulate the contents of any transmissions. Once the data reaches the server, we consider the system behaviour to be *semi-trusted*. By this we mean that the server, database, and verifier are expected to honestly follow Gringotts protocols, but are not trusted to authenticate or manipulate data. That is, a database administrator or user may choose to attempt to alter data stored on the database or passing through the server, but may not silently change the code of the Gringotts programs, since these can be checked against the official release. The data sources, which should be physically and remotely difficult to access, is considered trusted; so is the Time Stamp Authority, which is provided by a trusted third party.

Throughout this paper we consider data generator keys to be inviolate. The means of safely binding keys to entities is outside the scope of this paper, and remains an open problem. We also assume that an attacker is computationally bounded, meaning that she has finite time and computational power.

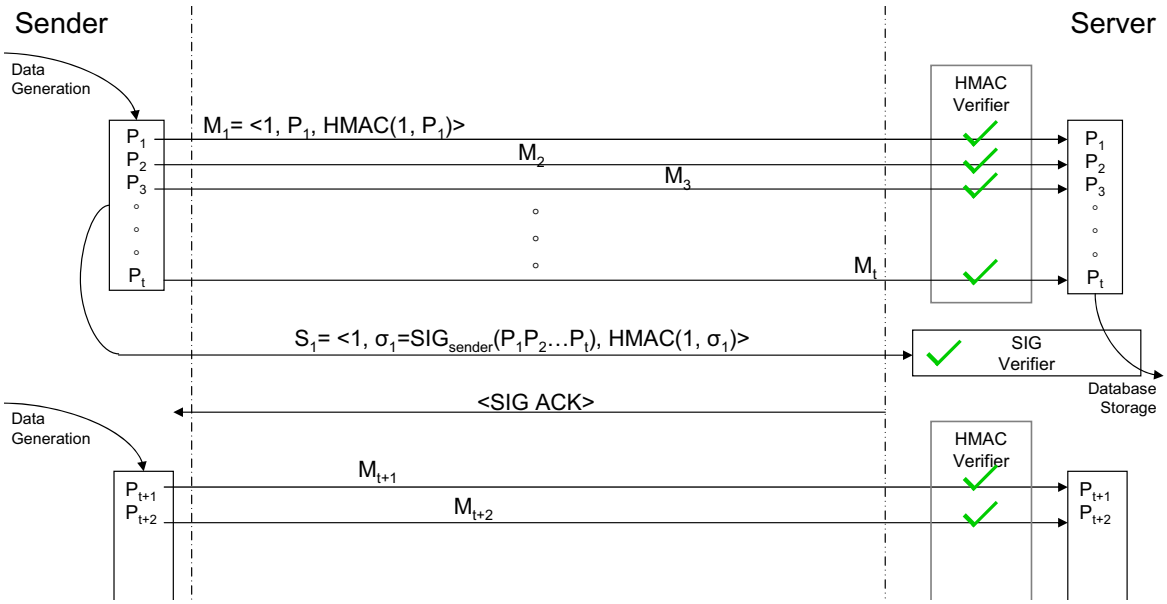

Fig. 2. Transmission of a block using the Gringotts Signature Scheme. Packets P are sent in blocks of t messages M, which include a packet number and an HMAC. After an entire block is sent, the block signature is sent and acknowledged, and a new block is begun.

B. Signature Scheme

A digital signature only guarantees data integrity as far back as the device that created the signature, so the Gringotts System requires that signatures be generated by the same entity that generated the data, e.g. in a surveillance system, signatures must be created by the security cameras. Since the network is completely untrusted, data must be signed before it is sent to the server. From micro-benchmarking experiments, we determined that signing each packet as it is transmitted is prohibitively slow. As discussed in §II, an examination of existing signature schemes for streamed data did not yield any scheme that maintained the integrity of the data stream while still transmitting fast enough to be practical.

Fig. 2 outlines the Gringotts Signature Scheme. As with more permissive streaming signature schemes [7], [11], Gringotts breaks the possibly infinite stream of data into finite blocks of t packets each for processing. The sender begins the protocol by sending a block of t messages. A message contains a data packet, an incrementing packet number, and Hash-based Message Authentication Code (HMAC) computed on the concatenation of the data and packet number. Generation of the shared key k used in the HMAC is system-dependent, but the handshake used in our implementation is described briefly in §V-A. Since the semi-trusted server can forge an HMAC using the shared key k, the HMACs are not used to authenticate data. However, the block size t must be less than the ENISA/NIST-required number of times an HMAC key can be securely used [18], and k should be periodically rolled over after a predetermined number of uses to disallow replay attacks.

Once a complete data block as been transmitted, the sender generates a signature for the entire block's data, and sends it to the server. The HMACs and packet numbers are not included in the signed block data. The block signature is sent with the block number, and the transmission is authenticated with an HMAC. Because the signature is generated by the camera, this scheme retains the property of non-repudiation of origin. The block signature authenticates the data to the server, and is later used by the Gringotts Verifier to provide authentication to the end user when that data is accessed.

If a data packet is corrupted, not only is that packet unverifiable, but the block it is part of also becomes unverifiable, since the final block signature will only be verifiable on the receipt of the entire, correct data block. This is the reason for including HMACs in our design. If the authentication process only relied on the final signature, then a single corrupted bit (either random or malicious) would require that the entire block be either dropped or resent. If the server cannot verify the signature and one of the packets in the current block has an unverifiable HMAC, then only the unverified packet needs to be resent for the server to be able to verify the signature.

If the server can verify the block signature, then it sends an acknowledgement of the verified block to the sender, and writes the block data and its associated signature to the database, which is described in §III-C. If the server cannot verify a block signature, instead of an acknowledgement it sends a reply to the sender requesting the corrupted messages or signature be resent, using the HMACs to identify the problematic transmissions. When the server has received the necessary messages and has properly verified the block signature, it acknowledges for the now-verified block to the sender. After a proper acknowledgement has been transmitted, the sender and receiver then proceed to the next block.

For practical systems, the number of permissible resend attempts must be bounded by a timeout. Infinite retry attempts would create a new vector for denial of service attacks.

C. Storage

Once a data block's signature has been verified, the next requirement for the Gringotts System is to preserve that non-repudiable signature so that at any point in the future, the data can be trusted without reference to who has had access to the server or the database. For the purpose, we format the database using the ERS protocol, which is designed to safely and verifiably store long-term archival data while maintaining provable data integrity [5].

If data needs to be stored for months or years, traditional proofs of integrity, such as digital signatures, can decrease in usefulness as their underlying cryptographic primitives or key sizes become outdated. Data protected by obsolete schemes must be resigned, and the resigning must be timestamped to prove that it happened before the signature was easily forgeable. However, an archival system may contain a multitude of signed data objects. In the case of security camera footage, even a few sensitive hours can require gigabytes of data and, if processed using the Gringotts Signature Scheme, may include thousands of signed blocks. As such, manually discovering and resigning all files is impractical.

A database implementing ERS stores its documents in a hash tree. A hash tree is built by concatenating and rehashing file hashes, with the root hash as its apex [19]. In ERS, the root hash is signed with a certificate from a Time Stamp Authority (TSA), a trusted third party that certifies the existence of an object at a particular time. Since a data file cannot be altered without altering the root hash and thus making the TSA certificate unverifiable, a verifiable root hash with a verifiable TSA certificate can be used to prove the integrity and existence of all the data in the hash tree from the time the certificate was generated. The data in an ERS tree, once made, is effectively immutable. For Gringotts, which receives data in infinite streams, maintaining a single ERS tree is impossible, so the server periodically saves accumulated data as a new ERS tree. If the ERS cryptographic functions need to be updated (if they become old or insecure), all existing ERS trees are gathered into a single updated tree, which then receives the new TSA certificate for its root hash.

ERS as defined in [5] offers no guarantee on the order of the files in a hash tree (although order between hash trees may be guessed by the history of the TSA certificates, which include timestamps). While not all data that may be used as evidence is ordered, in cases such as security camera recordings or system logs, order is essential. For that reason, our Gringotts implementation uses Ordered ERS, a modified version of ERS, described in [17]. In Ordered ERS, two types of management files are added. *Registry* files list the order of the data files from a given stream in a given hash tree. A single hash tree may have multiple registry files (for multiple streams), and a single stream will have multiple registry files (in different ERS trees). Registry files also contain a pointer to the previous registry file for the stream they list, so that the order for registry files (and thus the order for the ordered groups of data files they describe) can be determined from this linked list. The second management file specific to Ordered ERS is the *terminator* file. There is one terminator file per stream, and it contains a pointer to most recent registry file. The terminator file is used to find the end of the ordered list of files, either for appending a new data group (and its registry file) or for finding and verifying an existing data file. Fig. 3 shows two data streams stored in Ordered ERS hash trees.

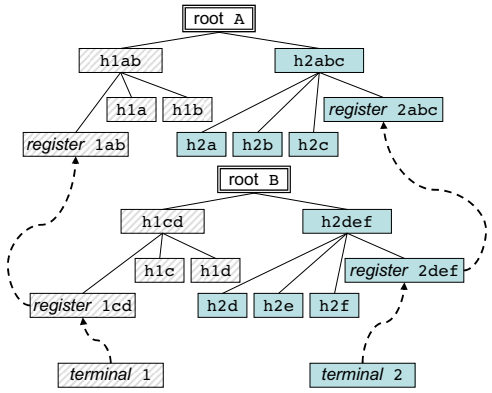

Fig. 3. A set of Ordered ERS hash trees. Two streams, 1 (in grey stripes) and 2 (in solid blue), are stored in two hash trees. The upper tree (with root A) covers data files a and b of Stream 1 and data files a, b, and c of Stream 2. The lower tree (root B) covers data files c and d for Stream 1, and data files d, e, and f for Stream 2. Data file hashes (the leaves of the trees) are labeled by stream name and data block. For example, "h2b" is the hash of data file b in Stream 2. Intermediate hashes are computed by concatenating the node's children and hashing the result. For example, "h1cd" is the hash of the register file for stream 1, files c and d, concatenated with the hash of data file c in Stream 1 and the hash of data file d in Stream 1. Pointers between management files are marked with dotted arrows.

D. Verification

The Gringotts Database is an access controlled, read-only data store. The exact details of the access control policy are outside of the scope of this project, but can be implemented using the database's native access control policies, or third-party software. Even if a user has permission to access data stored by Gringotts, that data must be verified before being made available. The Gringotts Verifier first identifies the Evidence Record for a requested data file, and verifies the Evidence Record's hash tree and TSA certificate. Each Evidence Record contains the data file and its reduced hash tree, so a single evidence record is sufficient to verify that file. Assuming that Ordered ERS is used, the Verifier also verifies the validity of the data ordering from the the metadata. Register files list the order of the data group they describe (their sibling data files in the hash tree). Since register files are included in the Ordered ERS hash tree, a verifiable root hash means that the register files for that tree are also valid, and the ordering they provide can be used to verify that no files have been deleted, and all requested files remain in the correct order. Finally, the Verifier checks the data signature (the block signature from the sender). Once the signature and ERS tree have been verified, the data can be returned to the user.

IV. Security Analysis

The goal of the Gringotts System is to protect data in such a way that it can be trusted by a court or other third-party to accurately represent its purported contents. We trust data generators, such as security sensors or cameras entirely, and we consider the server and database semi-trusted: we assume that they will be maintained according to the protocol, but

not that the stored data is immune to tampering. We expect that the server will accept and store received data, and that the database will be updated if the cryptographic algorithms used by ERS become obsolete. Since we are concerned only with the integrity of the data which is stored in the database, a misbehaving server that deletes or ignores incoming data is outside of the scope of this project. A malicious administrator could neglect or attempt to obfuscate ERS maintenance, but the ERS protocol includes a trusted third party-provided timestamped certificate, so the attacker would be unable to silently disrupt the system operations. The network is considered to be completely untrusted.

In §III-A, we introduced the threat model Gringotts is designed to protect against. In this section we discuss the security of the various parts of the Gringotts System. Since the network is untrusted, we address the possibility of attacks on the transmission of data from the sender to the server (also referred to as the receiver in this section to avoid confusion). Considering the semi-trusted database and server, we consider first outsider attacks (primarily on the database), and then the possibility of a malicious party who has direct access to some part of the system.

A. Network Security

We first consider an attack on the data as it is transmitted over the network. Out implementation of the Gringotts Signature Scheme uses ECDSA, and this proof assumes the use of elliptic curve (EC) cryptography for both block signatures and the computation of the HMAC keys. However, in the case where ECDSA provides insufficient security guarantees, it may be replaced by a different unforgeable under chosen message attack (UF-CMA) signature scheme, and a similar proof should be constructible.

We assume that ECDSA is existentially unforgeable (UF-CMA) by either the receiver or a third party, and that an HMAC is existentially unforgeable by a third party. Since HMACs are based on a shared private key, obviously there is no way to prevent the receiver from forging an HMAC. We also assume that the Elliptic Curve Diffie Hellman (ECDH) Problem is hard (unsolvable in polynomial time).

Based on these assumptions, and considered independently, the HMACs and block signatures must be existentially unforgeable. This means that an adversary could not, without detection, alter existing packets or add packets to the stream. To silently corrupt or insert a packet, the adversary would have to successfully forge both the packet's HMAC and the block's signature, which would contradict our assumptions. An adversary also cannot cause messages to be dropped. The sender and receiver can communicate using TCP, where non-malicious dropped packets would be dealt with on the transport layer. However, even if the adversary forged all following TCP headers to ensure that the dropped packet was not acknowledged at the transport layer, application-level authentication ensures that dropped packets are identifiable. Since the messages include the packet number, dropping a packet and altering the following packet number is impossible without forging both that following packet's HMAC, as well as, eventually, the block signature. These forgeries would contradict our assumptions. If an attacker attempts to drop an entire block, the same contradiction applies on a block scale, as the block signatures are similarly numbered, and that number is also authenticated by an HMAC.

The server is unable to alter the data received from the senders, since to do so would require the forgery of a block signature. While the server can forge any HMAC, the HMACs are only used to ensure the data integrity of the packets travelling over the network. The server does have the power to drop blocks, although not silently for timestamped data. If the data covered by the block signature includes information about data or transmission times or ordering, missing blocks will be visible. If an attacker on the server writes an unverifiable block to the database, this error will become apparent during retrieval, since the verification process checks the validity of both the ERS tree and the block signature.

In the Gringotts System, the ECDSA signatures and the shared HMAC key k (assumed to be computed using a ECDH key exchange) are not completely independent – they can (and in our proof of concept do) both rely on the same set of EC parameters and public-private key pairs belonging to the sender (S) and the receiver (R). This relationship, however, does not reduce the security guarantees previously discussed. Let us call the sender's secret key, d_S, and her public key, $Q_S = Gd_S$; and the receiver's secret key, d_R, and her public key, $Q_R = Gd_R$ (where G is the publicly available generator for the group).

Suppose, using some algorithm F, one could forge a ECDSA signature from reading all the HMACs exchanged between the two hosts. Then, without loss of generality, the receiver R could forge the sender S's signature as follows. R determines k by multiplying her private key with S's public key (Q_S), and given k she can create (i.e., read) as many HMACs are required for input to F. R then uses F to forge a message-signature pair purporting to come from S. Thus, R will have forged S's signature with no more than Q_S, which is publicly available. This clearly contradicts our assumption regarding the security of ECDSA, so we can see that knowledge of any number of HMACs sharing the same EC parameters does not allow R or a third-party to forge an ECDSA signature.

Since Gringotts' security guarantees derive from the ECDSA block signature, it is sufficient to show that using an ECDH key k based on the same EC parameters as the signature does not compromise the signature. The converse (that a third party seeing the ECDSA signatures cannot compromise future HMACs based on the same key) is not a security concern for the system, merely a performance one. Forged HMACs mean that a unverified block must be completely resent, but do not allow the attacker to corrupt a block stored in the database. While we do not formally prove that forged HMACs are impossible, we suggest the following intuition: If we could gain information about the HMACs from ECDSA signatures, then either we have gained information about key k (which contradicts our assumption of the security of ECDH), or about the information required to compute key d_S or key d_R, namely the secret keys of the communicating hosts (which contradicts out assumption that ECDSA is unforgeable).

Replay Attacks: Since the block and packet numbers are integral to the prevention of dropped packets, and these numbers must, in a real world implementation, wrap around, we consider the possibility of replay attacks. Replaying only

packets has little effect on security, as the new block signature would fail to verify if the old block data were sent instead. Replaying an entire old block complete with its old signature is a possibility. However, the key k used in the HMAC is regularly rolled over to avoid brute force attacks on the security of the HMACs. In our current deployment, that rollover occurs before the integers wrap around (k should be reset around every 2^{20} transmissions [18]; the integers wrap around every 2^{32} transmissions), so it is impossible for an attacker to replay an old block with the correct new HMACs. In a deployment on different hardware, the size of an integer must be considered. If the integers wrap around before k is required to roll over, that implementation should roll k over earlier to prevent the development of a vector for replay attacks.

B. Database Security

Each Evidence Record in the database is stored with its signature and is part of a timestamped Ordered ERS hash tree. Thus the data existence is provably verifiable, as described in [5], §III-D, §IV-A. The ordering of the evidence records, however, is entirely maintained by the Ordered ERS registry files, which are generated by the server, and thus cannot be authenticated by the data source. Given the semi-trusted nature of the server and the database, we cannot assume that either will reliably authenticate data ordering. However, for ordered data, the records themselves are likely to contain some ordering information. If so, these data timestamps may not be aligned with the server's clock, but they should be internally consistent, and thus provide reliable confirmation of the server-generated ordering.

The timestamp on an ERS hash tree is generated when the hash tree is created. Since the senders' clocks are not synchronized, and hash trees are only generated periodically, the hash tree timestamps have no actual relation to when the data was generated.

C. Insider Attacks

Since a motivation for the Gringotts System is the admissibility of data as digital evidence in court cases, and the owner of data may not be disinterested in the results of said case, it is essential that we also consider the danger of internal tampering. We consider two types of internal threat: those that threaten the semi-trusted portions of our system, and those that threaten the trusted components.

1) Semi-trusted: Physical access to the database or the server would allow an attacker to delete or corrupt stored data, but not to silently alter it. Since the signatures come from a data source, and we assume that those signatures are unforgeable (see §IV-A), an attacker targeting the server or the database will not be able to provide a valid signature for the altered data, and the forgery would be detected when a user attempted to the data.

2) Trusted: Insider attacks on trusted components are outside of the scope of Gringotts' threat model. However, in reality, this may not be the case, so we briefly outline the dangers of such attacks, and possible safeguards. If an attacker has physical access to a data source, she could alter the data without being detected. In the case of security sensor data (e.g., surveillance cameras), we assume that the data generation hardware is located in protected or hard to reach places. Should physical access to data senders be a concern, physical safeguards such as tamper-evidence hardware can also be used. If a data source is compromised, or the its cryptographic key pair is tampered with, rekeying will require careful auditing to maintain the trusted nature of the data sources. If an insider attacks the TSA, the timing and security of root hash certificates are suspect. TSA providers generally have stringent access control policies when it comes to their servers, however.

V. Implementation and Performance

Our proof of concept Gringotts implementation uses a Mac mini with a 2.5GHz Intel Core i5 processor and 4GB of 1333MHz DDR3 memory running OS X 10.8 as the server and database, and a third generation iPad with a 1GHz ARM Cortex A9 processor and 1GB of 1024MHz DDR2 memory running iOS6 as the data source. The data generated for our performance analysis was video data encoded using MPEG-4 with a 640x480 frame size, and was transmitted at 15 frames per second. With the exception of the encoding type, this is within the range of standard industry security camera data transmissions. The difference in encoding is due to the proprietary nature of the standard industry encoding style, H.264. Like H.264, MPEG-4 uses variable frame sizes and types, so we believe that our signature scheme could be applied to an H.264 stream with similar overheads.

The data generation code consists of ˜2400 lines of Objective C and C code, and uses OpenSSL and FFmpeg libraries as well as CocoaTouch frameworks. The server code consists of ˜2700 lines of Java code, and also uses OpenSSL and FFmpeg libraries, as well as an implementation of OrderedERS. The FFmpeg URLProtocol that implements the Gringotts Signature Scheme is used by both sender and server code, and consists of ˜1200 lines of C code. These counts include some embedded unit tests, as well as trace and error messages.

The implementation of Ordered ERS used in Gringotts was created specifically for that purpose, and consists of an ERS implementation and ˜1500 lines of Java code. To the best of our knowledge, our proof of concept represents the first implementation of Ordered ERS.

A. Handshake

In §III-B, we presupposed the existence of a secret key k shared between the sender and the server. In our implementation, we include the following handshake, which allows the server to authenticate incoming data streams, and provides a protocol for computing the shared key k.

We assume that the server listens on a publicly known port for new connections. When it starts to run Gringotts code, each data source is assigned an EC key pair. The server is aware of each sender's unique public key information using some form of out-of-band authentication or trusted third party certification. This prevents unknown cameras from being added to the system.

A new data source initiates the handshake by sending its public key and a signed, session-unique stream name to the server's open public port. Our Ordered ERS implementation

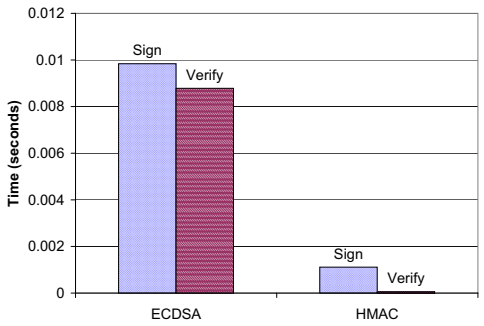

Fig. 4. Signature Scheme Timing. The signature generation and verification time required for ECDSA and HMAC, the two authentication algorithms used in the Gringotts System. ECDSA, while the fastest signature algorithm in general use, is still too slow to sign every packet for high-bandwidth data transmission, like video.

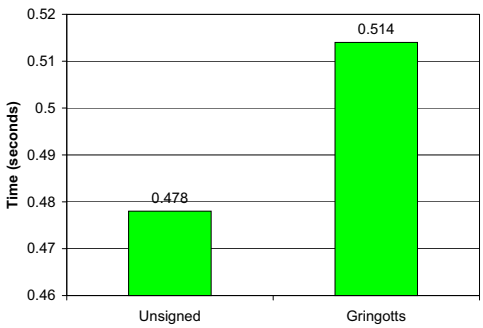

Fig. 5. Server Processing Time. The average time required for the server to process a 5s video sent from a data source, with and without the Gringotts Signature Scheme.

uses the stream names to differentiate ordered file groups from different sources. On receiving the initial transmission, the server confirms that the public key it has just received is one of its known data sender public keys. This check prevents an attacker from generating a new key pair and masquerading as a valid data source. The server also verifies the signature on the proposed stream name. A verifiable signature proves that the connecting entity is the data sender it purports to be, since no one else has access to the secret key associated with its public key, and thus could not produce a valid signature for that key.

Once it has authenticated the connecting data generator, the server selects an unused port, and starts a new thread to process data input from the new stream. The new thread sends the data generator its port number, and the sender initiates a Diffie Hellman key exchange. Once the exchange is finished, both the sender and the server have calculated a shared secret key k, and the sender begins sending data using the Gringotts Signature Scheme. The server stores verified data in the database, which runs Ordered ERS.

B. Signature Scheme

Fig. 4 shows the processing times required for ECDSA and HMAC generation and verification. The values shown were computed using the Objective C class *NSDate*. These signature scheme microbenchmarks were helpful in allowing us to determined the necessity of the Gringotts Signature Scheme. An analysis of video data produced during our experiments showed that the median data packet size was around 1000B, and that our experimental data source had a throughput of ˜0.18Mbps. Thus, we can see that our experimental data generator sent around 22 packets per second ($\frac{0.18Mbits}{1s} \times \frac{10^6bits}{1Mbits} \times \frac{1Bytes}{8bits} \times \frac{1packet}{1000Bytes}$). The experimental packet transmission time was ˜0.045sec. From Fig. 4, we see that using an ECDSA signature on every packet would increase the transmission time by 22%, and the verification time by 20%. An HMAC per packet, by comparison, increases transmission time by 2.4%, and verification time by only 0.12%. Using the Gringotts Signature Scheme, several hundred packets can be transmitted with only the overhead from HMAC processing, before a single ECDSA signature is required. How necessary the use of the Gringotts Signature Scheme is depends on the type of data being transmitted, and the network specifications. However, for large data transfers, such as surveillance video, network bandwidth is invariably a concern, and a 20% overhead for transmission is impractical.

Fig. 5 shows the overhead caused by the Gringotts Signature Scheme. The left bar ("Unsigned") shows the time required for the server to receive 5s of video data from the sender using only standard TCP with no additional protection. The right bar ("Gringotts") shows the time required for the server to receive 5s of video data from the sender using the Gringotts Signature Scheme over TCP. The server processing time was computed using the Unix *time* command on the server. The processing time measured for Gringotts includes the time required to receive additional data (HMACs and signatures), to verify HMACs and block signatures, and to send application-level acknowledgements. For our 5s video, the Gringotts Signature Scheme required an average of 0.514s processing time, as compared to the average 0.478s required to process 5s of video without signatures. Thus the overhead for using Gringotts is only 7.5% as compared to an unprotected stream. The number of packets per block t for these benchmarking experiments ranged from 100 to 500 without a statistically significant variation in timing.

For a larger scale experiment, we set up a small network over which two cameras recorded and transmitted data to a server running Gringotts continuously for about an hour without observing any disruption of transmission speed or quality. In comparison, the naive proposal to sign every packet with an ECDSA signature led to a stalled transmission from a single camera in under 5 minutes.

C. Storage

In our experiments, ECDSA signatures were of size 76B. The average packet size was around 20kB, meaning that the Gringotts block signatures increased the data storage by 0.38%. The ERS hash tree files required an additional 1.1%, and the Ordered ERS metadata, an additional 4.7% of disk space. Overall, Gringotts required 6.2% more storage than a system without integrity guarantees.

The current ERS library Gringotts uses is a simple Java implementation of the standard, which includes the local file system as its database. As such, it has no indexing, or

storage and processing optimization. Using an Ordered ERS implementation that runs on a well-optimized database could improve both storage space and verification time requirements. Similarly, more efficient storage of Ordered ERS metadata, much of which is currently stored in text and xml files, would decrease the required space.

VI. Conclusion

In an age where legal and contractual evidence is increasingly digital, it is necessary to be able to both prove the veracity of stored data, and to be able to trust data provided by others. In this paper, we have described the Gringotts System, which aims to indefinitely protect archival data. Using our novel Gringotts Signature Scheme and the existing ERS protocol to preserve a data source-generated signature over an untrusted network and for an unspecified period of time, this system should be applicable to data that must be stored securely for months or even years. We have also demonstrated the practical applicability of our design by implementing a proof of concept for the transmission of video data from multiple sources. In our implementation, the transmission time was increased by only 7.5% and the storage space by 6.2%, in comparison to an unprotected system.

While the Gringotts System as it stands fulfills our security goals, the database used in our implementation was unoptimized. For a system like Gringotts, which expects to collect large amounts of data over long periods of time, a more efficient data store would be preferable. In future, we hope to implement the Gringotts Database for a distributed database management system.

Another direction for future research is the relation between a data source and its key pair, and how to securely communicate that information safely to the server. We are also interested in creating a secure method for renewing data source keys that may have become compromised.

Comparing data stored on Gringotts is also an open question. As the current design stands, timestamps are provided by independent data sources, with no guarantee of clock synchronization. Thus, we also hope to develop a clock synchronization method for distributed data generators.

This paper has used video surveillance as a motivating example of the Gringotts System, and our proof of concept reflects that. However, we believe that our system can be applied to a variety of data types and sources with equal efficiency. The Gringotts Signature Scheme can also be used separately from the archival storage system for any project needs a complete, and completely authenticated, data stream.

Acknowledgments

The authors would like to thank Mr. Shimaoka and Mr. Sato from Secom's Intelligent Systems Laboratory for their advice and assistance on this project. We are also indebted to JK Rowling for the name of our system, which is borrowed from the bank in her *Harry Potter* series – the safest place in the wizarding world to keep anything you want to protect.

References

[1] C. Initiative. (2006) The Admissibility of Electronic Evidence in Court: Fighting Against High-Tech Crime. [Online]. Available: http://www.coe.int/t/dghl/cooperation/economiccrime/cybercrime/cy%20activity%20bul/cy%20activity%20bul%20cyberx%20libro_aeec_en.pdf

[2] (2013) CCTV Tipping Point Still Coming in 2014: New Video Surveillance Gear Sales to Surpass Analog Next Year. IMS Research. [Online]. Available: http://www.imsresearch.com/news-events/press-template.php?pr_id=3643

[3] J. Honovich. (2012) How to Design a Video Surveillance Solution. [Online]. Available: http://ipvm.com/report/how_to_design_video_surveillance_solution

[4] T. Connor. (2011) Peep show: inside the world of unsecured IP security cameras. [Online]. Available: http://arstechnica.com/gadgets/2011/01/one-mans-journey-through-the-world-of-unsecured-ip-surveillance-cams

[5] T. I. Trust. (2007) Evidence Record Syntax. Request for Comments 4998. [Online]. Available: http://www.ietf.org/rfc/rfc4998.txt

[6] R. Gennaro and P. Rohatgi, "How to Sign Digital Streams," in *Advances in Cryptology (CRYPTO'97)*. Springer, 1997, pp. 180–197.

[7] C. K. Wong and S. S. Lam, "Digital Signatures for Flows and Multicasts," in *Network Protocols, 1998. Proceedings. Sixth International Conference on*. IEEE, 1998, pp. 198–209.

[8] Y.-h. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast (keynote address)," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 28, no. 1. ACM, 2000, pp. 1–12.

[9] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient Authentication and Signing of Multicast Streams over Lossy Channels," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*. IEEE, 2000, pp. 56–73.

[10] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Efficient and Secure Source Authentication for Multicast," in *Network and Distributed System Security Symposium, NDSS*, vol. 1, 2001, pp. 35–46.

[11] P. Golle and N. Modadugu, "Authenticating Streamed Data in the Presence of Random Packet Loss," in *Network and Distributed System Security Symposium (NDSS)*, vol. 1. ISOC, 2001, pp. 13–22.

[12] J. M. Park, E. K. Chong, and H. J. Siegel, "Efficient Multicast Stream Authentication using Erasure Codes," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 258–285, 2003.

[13] C. Tartary, H. Wang, and S. Ling, "Authentication of Digital Streams," *IEEE Transactions on Information Theory*, vol. 57, no. 9, pp. 6285–6303, 2011.

[14] M. Franklin, R. Gelles, R. Ostrovsky, and L. J. Schulman, "Optimal Coding for Streaming Authentication and Interactive Communication," in *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 19, 2012, p. 104.

[15] ETSI. Quantum Key Distribution. 650 Route des Lucioles, 06921 Sophia Antipolis, France. Accessed March 31, 2014. [Online]. Available: http://www.etsi.org/images/files/ETSITechnologyLeaflets/QuantumKeyDistribution.pdf

[16] Long-Term Archive and Notary Services Working Group (concluded). Accessed January 29, 2014. [Online]. Available: http://datatracker.ietf.org/wg/ltans/charter/

[17] H. Date and M. Sato, "Verifiable data ordering in the ers standard timestamp-granting digital signature method," Japanese Patent 2013-46 288 A, 3 4, 2013.

[18] *The Keyed-Hash Message Authentication Code (HMAC)*, Information Technology Laboratory Std., 2008.

[19] R. C. Merkle, "Protocols for Public Key Cryptosystems," in *IEEE Symposium on Security and privacy*, vol. 1109, 1980, pp. 122–134.