WebTrust – A Comprehensive Authenticity and Integrity Framework for HTTP

Michael Backes[1], Rainer W. Gerling[2], Sebastian Gerling[1], Stefan Nürnberger[1], Dominique Schröder[1], and Mark Simkin[1]

[1] CISPA, Saarland University
[2] University of Applied Sciences Munich

Abstract HTTPS is *the* standard for confidential and integrity-protected communication on the Web. However, it authenticates the server, not its content. We present WebTrust, the first comprehensive authenticity and integrity framework that allows on-the-fly verification of static, dynamic, and real-time streamed Web content from untrusted servers. Our framework seamlessly integrates into HTTP and allows to validate streamed content progressively at arrival. Our performance results demonstrate both the practicality and efficiency of our approach.

Keywords: HTTP, Integrity, Authenticity, Verifiable Data Streaming

1 Introduction

The *Hypertext Transfer Protocol* (HTTP) is the standard protocol in the World Wide Web that allows clients to request and receive any type of content from a server such as static, dynamically created, or even live streamed content [9]. HTTP is a pure transfer protocol that does not provide any state information or security guarantees by itself. For many applications, however, security guarantees are strictly necessary and various extensions haven been proposed. The standard protocol to provide security guarantees is HTTPS (HTTP over TLS [31]), which establishes a secure channel between the client and the server. Although a secure channel guarantees that the transferred content has not been modified during the transmission, it *neither* guarantees the authenticity *nor* the integrity of the delivered document itself. Moreover, if an attacker gained access to the server, any content the attacker would put on the server would be authenticated at the client's side, since HTTPS only authenticates the connection. This situation is completely unsatisfactory, since one can neither rely on the information in published documents nor prove their correctness to third parties (non-repudiation). Consider for example news aggregators that mainly serve information generated by news agencies: An established HTTPS connection to a news aggregator or social network cannot guarantee anything about the authenticity and integrity of the delivered content itself. The problem becomes even more challenging when we want to ensure the authenticity and integrity of live streamed content.

One intuitive approach to ensure the authenticity and integrity of content would be to append a digital signature. However, signatures need to be downloaded separately and there is no unified solution that integrates into the existing infrastructure. Moreover, in case of large files, a single signature can only be verified after the content has been downloaded completely. Individual signatures also do not fulfill the requirements defined by today's Web resources that embed other resources such as images and scripts. Especially for (live) streamed content, this calls for a flexible and efficient solution that allows to verify content on-the-fly. In order to prevent an attacker from being able to replace partial content with other, also signed, content, all relevant interconnected documents need to be verified together. Mobile broadband providers replace embedded pictures with their compressed versions in order to save bandwidth [10]. Certain Internet providers even go as far as to inject advertisements into foreign websites [30].

1.1 Contribution

In this paper we present WebTrust, a comprehensive integrity and authenticity framework for static, dynamic, and live streamed Web content that seamlessly integrates into the existing infrastructure. Our framework allows content generators to publish authenticity- and integrity-protected content (from now on referred to as *WebTrust protected content*) on untrusted servers. In addition, WebTrust offers protection against active network attackers. The integrity and authenticity of downloaded HTTP documents can further be proven to third parties in an offline setting (non-repudiation). The verification of documents takes place on-the-fly (progressive content verification (PCV)) while the document is still being downloaded. In particular, WebTrust enables the client to detect any modified data packet upon arrival without downloading the entire document. Our solution adapts recent cryptographic primitives and profits from the lessons learned in previous approaches that focus on subsets of the aforementioned problems to realize our comprehensive integrity and authenticity framework [32,2,12,23,37,14]. WebTrust further supports efficient data updates and enables the usage of Web caches (the latter only, if confidentiality is not needed). Finally, our concept and implementation supports individual verifiability of content aggregated from different authors via IFrames.

1.2 Related Work

We discuss related frameworks that also provide authenticity and integrity guarantees and we compare them to WebTrust in Table 1. In that table, we compare the approaches w.r.t. the following features: The 1^{st} column indicates if the framework supports *verifiable authorship* meaning that the content can be verified against the author and not (only) against the server. The 2^{nd} column shows if documents can be *revoked*, i.e., the author can enforce and immediate expiration. The property of *non-repudiation* is compared in the 3^{rd} column and allows proving the authorship to third parties. The 4^{th} column indicates if the content can be *updated*, i.e., updating parts of the content is possible without

Table 1. Comparison of approaches to protect the integrity of HTTP transferred data

Feature	1	2	3	4	5	6	7
SHTTP [32]	–	–	–	–	–	S/D/L	–
HTTPS [31]	–	–	–	–	–	S/D/L	–
SSL Splitting [18,19]	–	–	–	–	✓	S/D/L	–
Bayardo and S. [2]	✓	–	✓	–	✓	S/D/–	✓
Sine [12]	–	–	✓	–	✓	S/D/–	✓
HTTPI [6]	–	–	–	–	✓	S/D/L	–
Spork [23]	–	–	✓	–	✓	S/D/–	–
HTTPi [37]	–	–	✓	–	✓	S/D/–	✓
iHTTP [14]	–	–	✓	–	✓	S/–/–	✓
WebTrust	✓	✓	✓	✓	✓	S/D/L	✓

Legend: 1. Verifiable Authorship 2. Document Revocation 3. Non-repudiation 4. Data updates 5. Caching/CDN-Support 6. Content types (**S**tatic, **D**ynamic, **L**ive Streaming) 7. Progressive Verification ✓: yes/full support, – no support.

re-signing the entire dataset. The 5^{th} column shows if caches resp. content distribution networks (CDN) are supported, i.e., the content can be distributed to different servers without harming the verifiability. In the 6^{th} column the different supported content types are listed such as static, dynamic, and streamed live content. Handling streamed content is particularly challenging as full precomputation is generally not possible. Static content is a mere copy of the file to the client, while dynamic content is generated on demand, and (live) streamed content is a stream of data that has an infinite size and is not known in advance. The 7^{th} column compares the approaches w.r.t. progressive verification meaning that the content can be verified while loading. Progressive verification is desired in setting where the clients do not want to wait until the end of a stream to verify any of the security properties. Due to space constrains we cannot discuss each approach in detail, but the chart already shows that none of the existing approaches provides a comprehensive solution to all common usage scenarios of HTTP. Further approaches exist that focus on efficient methods for a specific data type (cf. [20,8,29]), or focus on the data transmission of files on the Internet, or focus on streams such as [13,28] (not in real-time) or [27] for multicast streams over lossy channels with real-time support. Two less closely related approaches [11,30] also consider the problem of providing integrity for HTTP, however, they do not provide security guarantees in a cryptographic sense. Therefore we omit a more detailed discussion of these papers.

2 System Model

In the following we provide a high-level overview of WebTrust and discuss the attacker model as well as the underlying assumptions of our system. The global setup of WebTrust consists of three parties: the *client* with a Web browser, the HTTP-based Web *server*, and the *content generator* (cf. Figure 1).

Figure 1. WebTrust system overview

The content generator either creates WebTrust-protected documents a priori and uploads them to the Web server (static content), or the content is created on-the-fly by the content-generator and merely relayed by the server to the client (dynamic content). The client is then able to request protected resources based on the Unified Resource Identifier (URI) from the Web server and to progressively verify their integrity and authenticity with the help of the content generator's public key during the arrival. Depending on the scenario, the Web server can also fetch dynamic content from the content generator. The content generator and the Web server do not have to be different entities. However, we recommend them to be different whenever possible to mitigate the risks of key exposure through Web server breaches. Splitting these entities also allows us to assume an untrusted Web server which is accessible from the Web and potentially vulnerable.

2.1 Security Objectives

The goal of WebTrust is to provide robust security guarantees to users. In particular, our system needs to fulfill the following security objectives: *authenticity*, *integrity*, *validity*, and *freshness* of Web documents with respect to their author:

- **Authenticity** ensures that content indeed stems from the alleged author.
- **Integrity** ensures that content cannot be altered after its generation without being detected.
- **Freshness** ensures that a client always receives the *latest* version of a document, i.e., a man in the middle cannot replace a response with an integrity-protected and authentic, *but old* copy of a requested document.
- **Document revocation** ensures that a document was not actively revoked by its author.
- **Non repudiation** allows to proof the authenticity of documents to third parties.

Depending on the scenario, confidentiality of the transmission needs to be provided as well. This is not explicitly stated as an security objective, since this is orthogonal to our solution and can be achieved by transmitting WebTrust protected documents via HTTPS.

2.2 Attacker Model

We assume an active adversary that is able to eavesdrop and arbitrarily modify all network traffic. Such an adversary could be, for instance, the Internet service provider that is in control of the network connection. In addition, we assume that the adversary is able to fully compromise the Web servers in our

scenario (including the servers of a content distribution network (CDN)). This effectively grants the attacker access to all files stored on such server and allows to manipulate all requests and responses processed by them.

2.3 Assumptions

To provide robust security guarantees, it is central for WebTrust that the cryptographic keys used to sign content cannot be accessed by an attacker. Therefore, WebTrust has the following requirements to achieve its goals:

1. The content generator stores sensitive keys to sign content locally. We assume that these keys cannot be accessed by an attacker.
2. In case the content generator and the Web server are the same entity (implies that keys are stored locally on the Web server), we assume that the Web server cannot be compromised by an attacker. Otherwise, we consider the Web server untrusted without further assumptions.
3. We assume a standard trusted PKI that provides additional support for WebTrust content revocation lists (WT-CRLs) (cf. Section 4).

3 Theoretical Foundations

In the following we introduce the cryptographic primitives required by WebTrust, namely *elliptic curve cryptography* [22], *collision-resistant hash functions* [16], *chameleon hash functions* [17], *digital signatures* [16], and *verifiable data streaming* [34]. We use the following notation: By $y \leftarrow A(x)$ we denote the execution of a probabilistic polynomial time (PPT) algorithm on input x with output y.

3.1 Hash Functions

Collision-Resistant Hash Functions. We assume the existence of compressing collision-resistant hash functions. Roughly speaking, a function H is called collision-resistant if the probability that an efficient adversary finds two distinct pre-images $m_0 \neq m_1$ that map to the same image $H(m_0) = H(m_1)$ is negligible. WebTrust supports any state of the art collision-resistant hash function.

Chameleon Hash Functions. A chameleon hash (CH) function is similar to regular collision-resistant hash functions that are based on number theoretic assumptions and it provides additionally a trapdoor. Invertible chameleon hash functions are defined through the tuple $\mathcal{CH} = (\mathsf{chGen}, \mathsf{ch}, \mathsf{col}, \mathsf{scol})$ [17]. Its key generation algorithm $\mathsf{chGen}(1^\lambda)$ returns a key pair (sk_{ch}, pk_{ch}). The function $\mathsf{ch}(x; r)$ is parametrized by the public key pk_{ch} and outputs a hash value $h \in \{0,1\}^{\text{out}}$ for a input message $x \in \{0,1\}^{\text{in}}$ and a randomness $r \in \{0,1\}^\lambda$. The trapdoor sk_{ch} allows us to efficiently find collisions, i.e., a randomness r' such that both (x, r) and (x', r') will be mapped to the same hash value. CH functions can be instantiated from many number theoretic assumptions, such as the discrete logarithm assumption [17,1], the factoring assumption [35], and the RSA

assumption [1,15]. We use the scheme introduced by Nyberg and Rueppel [1]. Its security is proven in the generic group model assuming the hardness of some variant of the discrete logarithm problem in the cyclic group $\mathbb{Z}_p$. Our framework uses the elliptic curve variant of the Nyberg and Rueppel chameleon hash function. The advantage of using elliptic curves is that the chameleon hash values become smaller in size. For our choice of the curve, please refer to Section 7.

Merkle Trees. A Merkle Tree is a binary tree that allows an efficient verification of distinct elements from larger data sets. Data is stored in the tree's leafs and all inner nodes are computed recursively as the hash value of its concatenated children. The root node's value is published as the public key and can be used to verify each leaf individually. The authentication of a certain leaf requires all hash values that are adjacent to the nodes on the path from this leaf node to the root node, hence all proofs are logarithmic with respect to the amount of leafs.

3.2 Verifiable Data Streaming (VDS)

The verifiable data streaming (VDS) protocol [34] allows to authenticate data streams. VDS is based on a variant of Merkle Trees [21], so-called chameleon authentication trees (CAT), which have the following additional capabilities: New elements can be inserted into the tree without updating the root and already inserted elements can be updated efficiently. The correctness of each single element in the tree is publicly verifiable and can be proven to third parties. In essence, the security of VDS says that only the data owner can insert elements to and modify exiting elements in the CAT.

VDS Adaptation for WebTrust. The original VDS protocol [34] was designed to allow a computationally weak client to stream its entire data to a seemingly all-powerful server. In our scenario, the content generator is the client of the VDS setting. It streams content to an untrusted server, which is later received and publicly verified by other clients.

Consider the tree as depicted in Figure 2. The root $v_{1,0}$ of the tree is a hash, which is part of the content generator's public key. In the following, we denote the root value by ρ. Each left node of the tree is computed by a collision-resistant hash function and every right node is computed via a CH function. Since the CH function takes a randomness as additional input, it is necessary to store it in the right nodes. To verify a leaf in the tree, one has to compute an authentication path as in a traditional Merkle Tree (cf. Figure 3): To verify L_0 in the tree, the algorithm computes $v_{0,0} \leftarrow H(L_0 \| v_{-1,1})$ and checks if $\rho = H(v_{0,0} \| v_{0,1})$.

Now, let us assume that the client requests a video stream of a press conference and the client would like to verify the authenticity and integrity of the streamed content on-the-fly. The basic idea is to chop the stream in chunks of data such that a hash of each chunk is stored in a leaf. We illustrate this idea with a tree of small depth, but our data structure supports a binary tree of polynomial depth that can authenticate an exponential number of leaves. For an easier exposition of the main idea, we assume that the first two leaves L_0, L_1 are known in advance. To set up the tree, the algorithm picks two dummy

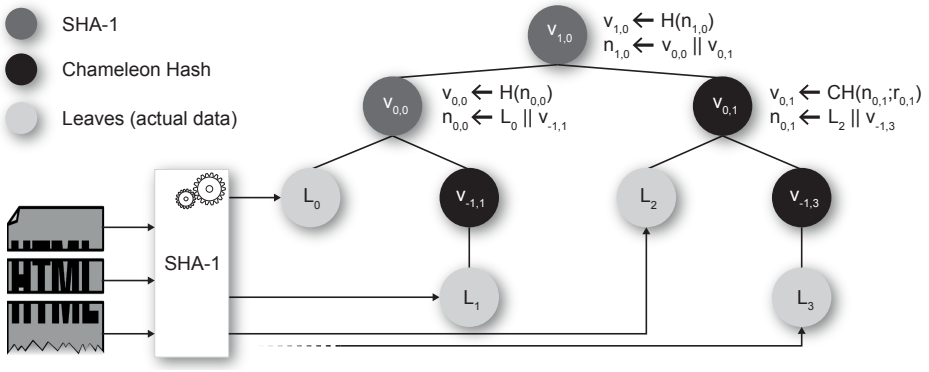

Figure 2. CAT for four leaves (L_0 to L_3) with vertices named $v_{height,index}$.

elements for the part of the stream that is unknown. In our case, it chooses elements $(n_{0,1}, r_{0,1})$ uniformly at random. The element $n_{0,1}$ is the input to the chameleon hash functions stored at node $v_{0,1}$ and $r_{0,1}$ is the corresponding randomness. Now, suppose that another element is streamed to the client that will be stored in the leave L_2. To add this elements to the tree, the server picks a dummy value for $v_{-1,3}$ and computes the collision with help of the algorithm $r'_{0,1} \leftarrow \mathsf{col}(sk_{ch}, n_{0,1}, r_{0,1}, (L_2 \| v_{-1,3}))$ and sends $(L_2, v_{-1,3}, r'_{0,1})$ to the client.

3.3 Digital Signature Schemes

Digital signature schemes allow to compute a signature σ on a document m using a private key sk, such that any party in possession of the corresponding public key pk can verify the validity of σ. Our construction requires a digital signature scheme that is secure against the standard notion of existential forgery under chosen message attacks [16]. WebTrust uses the RSA [33] signature scheme. It is provable secure in the random oracle model [3] and its underlying mathematical structure are composite order groups.

4 System Details

WebTrust leverages the previously described cryptographic primitives to achieve robust progressive integrity and authenticity verification of different content types with respect to their authors. In the following we describe our framework in detail and discuss how our security objectives (cf. Section 2.1) can be achieved. Moreover, we show that WebTrust is backwards compatible and supports caches as well as CDNs. WebTrust splits all content types into individual segments and processes them in the *signature provider* (Sign in the following figures) of the content generator (cf. Figure 4 for static content and Figure 5 for dynamic content). Our system currently supports two different signature providers, namely VDSECC and RSA-Chaining.

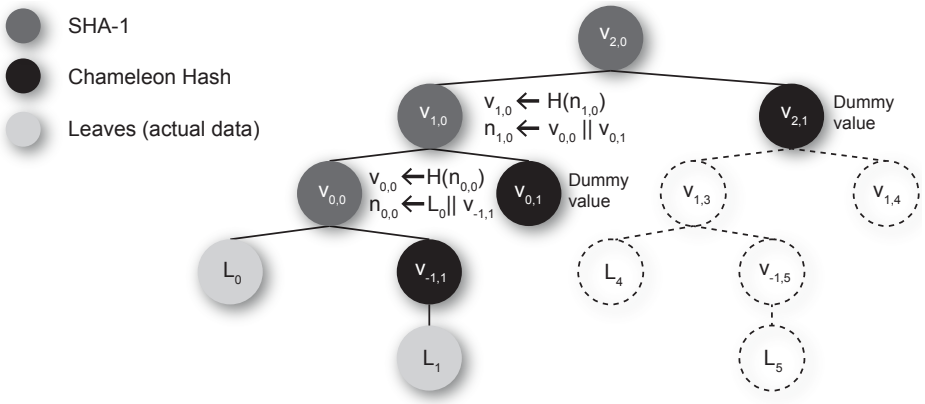

Figure 3. CAT of depth 5 that authenticates the leaves L_0 and L_1. Root node and left nodes are computed by a collision-resistant hash function, right nodes by a chameleon hash. The leaves $L_2, \ldots L_7$ are unknown. Appending the leaves L_4 and L_5 to the CAT (dotted in gray), requires the computation of a collision in nodes $v_{2,1}$ and $v_{-1,5}$.

VDSECC. VDSECC combines the VDS protocol with the elliptic curve variant of the Nyberg and Rueppel chameleon hash. It generates one CAT for each content object. For each content object its individual segments are hashed and added to the CAT. The first segment of a content object always includes meta-information such as the content URI, the creation date, and the expiration date. The ordering of segments is implicitly ensured through the CAT itself. Once a segment has been processed by the VDSECC signature provider, the returned proof is attached to the segment. The verification of each segment is done as previously described in Section 3.2.

RSA-Chaining. RSA-Chaining produces a signature chain by creating one signature for each pair of adjacent segments. For each incoming content object all

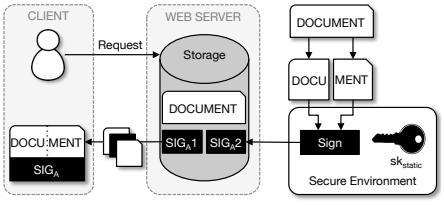

Figure 4. Static content: Data and signatures are broken into segments and are delivered interleaved.

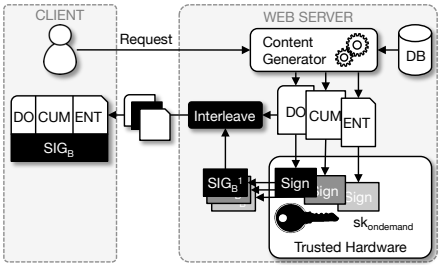

Figure 5. Dynamic content (Web server and content generator as a single entity): Signatures SIG_B^1, $SIG_B^2, \ldots$ are calculated during dynamic document creation.

its individual segments are first prepended by the same meta-information that we add to the first packet in the CAT and we additionally add the segments position in the chain. Afterwards, each segments is hashed and finally signed using RSA with the content generator's secret key. Once a segment has been processed by the RSA signature provider, each signature is attached to the segment. The client-side verification is based on a classical RSA signature verification against the content generator's public key. The chaining ensures that ordering of segments in the stream cannot be altered.

In order to achieve a seamless integration into the existing Web infrastructure, we now need to embed the signatures either created by the VDSECC or the RSA-Chaining signature provider into the data transmission in a backwards compatible manner. We achieve this by leveraging the existing HTTP/1.1 chunked mode. HTTP chunking ([9] section 3.6.1) is designed for documents whose size is unknown a priori and which are generated and transferred piecemeal to the client. Since HTTP chunking is a transfer encoding, it does not modify the content but merely the way it is transported to the client and thus perfectly meets our requirements. Clients that do not support WebTrust will simply ignore the attached signatures. Clients with WebTrust support will extract the embedded signatures for incoming segments, compute the hash of each segment, and finally verify whether the signatures are valid or not.

4.1 Progressive Content Processing

The progressive verification of WebTrust is enabled by splitting content into segments in combination with the particular design of the signature providers. Due to the content splitting, every segment is sent to the client with its own authenticity and integrity protection that can be verified immediately after arrival. Both signature providers are designed to allow the signing of a segment i without yet knowing the segment $i + 1$. VDSECC and RSA-Chaining allow the client to verify content object partially (e.g., media sub streams) without requiring the client to know the start, the end, or the file as a whole. WebTrust allows the progressive verification of all content types.

4.2 Individual Verifiability

Since WebTrust is supposed to be used by authors of content to protect their data, it allows to combine content of different authors in one website with individual verifiability. Our framework realizes this by loading each author's content into an individual IFrame of a website. Each IFrame triggers a separate WebTrust protected HTTP request. The user gets visual feedback about the verification result of each IFrame as shown in Figure 6.

In certain scenarios it may be desirable to ensure that an attacker cannot substitute any of the IFrames with a different signed resource. Consider the following scenario: A client requests the document `www.example.com/a.html`, which explicitly references a JavaScript file `www.example.com/script.js`. Assume that another script file signed by the same author with the same key exists

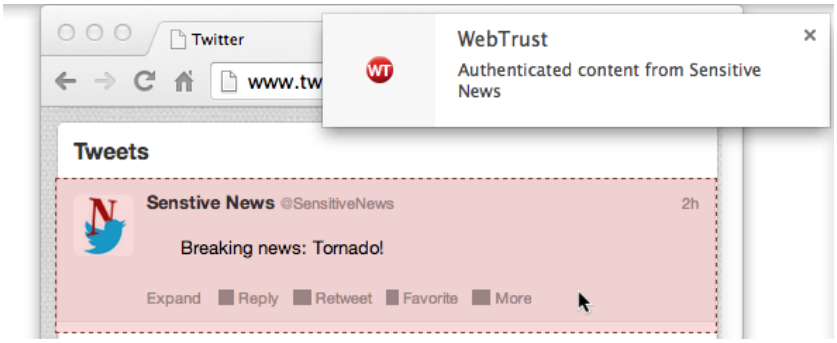

Figure 6. The WebTrust Chrome Extension showing the verified authorship of an embedded tweet in our modified Twitter page

at `www.example.com/anotherscript.js`. In this scenario, an attacker may replace `script.js` with `anotherscript.js` in the web response. To prevent this, WebTrust supports the incorporation of the content's full URI into the signature. This allows the detection of maliciously replaced files. Furthermore, WebTrust can be configured to include arbitrary HTTP headers in the signature. This may be particularly useful for critical headers such as cookies.

4.3 Content Updates

CATs allow content updates by design and our RSA signature chains can achieve the same functionality with the help of WT-CRLs. However, their update algorithms differ fundamentally. In particular, when using a CAT each update requires an update of the public key and involves updating a logarithmic amount of nodes in the tree. Instead, we introduce a second CAT which aggregates the roots of all the content object CATs as its leaf nodes. This method allows us to verify several content objects against only one public key, which is the root of the second CAT. When using RSA-Chaining, the update algorithm computes signatures for the new segment and adds the old segment to the WT-CRL. Hence, the size of the WT-CRL is linear with respect to the amount of updates.

4.4 Caching and CDN-Support

WebTrust content can be cached by proxies, cache servers, or CDNs, since the signature of static content is also static. Outsourcing content to CDNs is common practice to reduce load on a single server. In addition, CDNs equalize latencies around the globe by mirroring content at locations with a large distance to the central server. Dynamic content could also be cached from a technical point of view, but usually this content would have a `no-cache` directive set because caching does not make sense from a logical point of view.

4.5 Key Security

Content generators use secret keys to generate WebTrust protected content. These keys need to be protected against malicious access. In particular, in the case where the Web server and the content generator are the same entity, keys should be protected by a Hardware Security Module (HSM) as shown in Figure 5. With an HSM in place, an attacker that breaks into the Web server could still forge signed content by using the HSM as a signing oracle. However, he would never get hold of the key itself. This achieves the same level of security as storing HTTPS/TLS certificate private keys in an HSM.

5 Implementation

5.1 Server

We implemented the WebTrust server extension as a patch to the Apache Tomcat Web server 7.0.39. Our patch extends the existing processing routines for the HTTP chunked transfer encoding using a filter in the HTTP chunking driver of Tomcat. The HTTP 1.1 chunked mode allows so-called chunk-extensions that can be embedded into a chunk's header. The specification starts with the number of bytes in hexadecimal format, which can be followed by extensions of the format `;key=value,key=value,...`. The extensions are then followed by a line break before the actual chunk data. This overall format is repeated for every chunk. We attach the base64-encoded WebTrust protection as extension in the requested format (the example has a chunk size of 2761 bytes, `0xAC9` bytes in hex):

```
AC9;SIG=8CD3ABU8ULS2KMDN4HW3NK6A5BPP84HB6A7CC
```

Since the transmitted bytes are still well-formed HTTP, legacy clients will only extract the unmodified content. We successfully verified this compatibility of HTTP chunking extensions with Firefox, Chrome, Safari, Internet Explorer, wget, curl and Java. The public key for the overall verification is specified in the HTTP header itself, since it does not change for a single request. For that purpose, we added two additional HTTP headers to indicate the used WebTrust algorithm (here, VDS with SHA-1) and the corresponding public key:

```
Content-Verification-Scheme: 1.0/SHA1-VDSECC
Content-Verification-Key: 61KJHQ1J4NED97NBP2SJ44FP0
```

Similarly, chained hashes that are signed with RSA are implemented (`1.0/SHA1-RSA`).

The signature cache for static content is implemented using the `default` servlet of Tomcat. The `default` servlet is used when there is no dynamic servlet to generate content and the URI points to an actual file on the server. For each such file, we keep a list of chunk sizes and attached signatures. The `default` servlet implementation ensures that every response uses the same chunk size and hence can benefit from the stored signature chain. For the dynamic case, the implementation is slightly more complex: Servlets decide on their own how many bytes to flush to the network. For example, every time they call `flush()`,

the bytes written so far are sent to the WebTrust filter which takes care of accumulating them and eventually adding the signature. This ensures that if the servlet generates exactly the same output over two different runs, we generate exactly the same chunks. As those chunks appear to be static, we can cache them. The look-up procedure is realized using a hash map that is indexed by a tuple consisting of the SHA-1 hash of the content and the preceding signature.

To further reduce the server load we enable the WebTrust extension only if the client has requested its usage by sending the `'Accept-Content-Verification: SHA1-RSA'` header, where `SHA1-RSA` defines one of the supported schemes. Depending on the flag set by the client, the server responds using the requested scheme. The implementation of the cryptographic primitives on the server side is based on the `SunRsaSign` and `Sun` cryptography providers for non elliptic curve primitives as delivered with Oracle's Java [26], and based on Bouncycastle [4] for the elliptic curve primitives, and our own Java implementation for the CAT.

5.2 Client

We implemented the client-side prototype as a patch to the open source Chromium browser 29 [7] in combination with a browser extension. The patch targets the chunked-mode handling of Chromium and is used to parse and verify the WebTrust integrity and authenticity proof of incoming documents. Moreover, it includes the routines for adding the WebTrust headers into Web requests, which can be switched on and off. The required cryptographic primitives build upon the OpenSSL library [25] and include our own implementation for CATs in C++. The browser extension is used to prototype the UI for providing the user with feedback about the verification result. It also supports to give feedback for the individual verifiability of documents loaded inside of IFrames as described in Section 4.1. We would like to stress that this approach is merely used for prototyping the client application. A future release of the system is supposed to directly integrate the individual verifiability into Web browsers instead of an extension to optimize usability and performance. We would like to point out that although our concept leverages a PKI for freshness and revocation, our client-side prototype focuses primarily on the implementation and evaluation of our new signature providers and hence does not have an implementation for revocation.

6 Security Evaluation

In the following, we discuss how WebTrust fulfills the security objectives defined in Section 2.1 and how WebTrust is protected against attacks.

6.1 Integrity, Authenticity, and Non Repudiation

Our integrity check is based on hashing content segments with a collision resistant hash function. The property of collision resistance guarantees that an

attacker cannot find a second chunk that maps to the same hash value. To ensure that content segments cannot be replaced, all hashes are authenticated by one of the signature providers. Since the attacker can neither access the author's secret keys, nor forge valid signatures for RSA-Chaining or VDSECC without the secret key, the hash as proof of integrity cannot be replaced or modified. Since the proofs of integrity are verified against a specific user's public key, this immediately provides authenticity. Therefore, the integrity and authenticity of data is guaranteed and our signature providers allow to prove the correctness of the WebTrust protected segments to third parties (non repudiation). Whenever embedded content or a client-side script fetches another resource, this triggers another HTTP request, which is then also verified by WebTrust.

6.2 Freshness and Content Revocation

Our framework supports freshness, i.e. the user always obtains the latest version of the requested content object. This is achieved by incorporating an expiry date into the content object. If content is replaced before it is expired, the old version is revoked and does no longer verify successfully. In the case of RSA-Chaining this revocation is achieved by adding the old version to the WT-CRL at the PKI. This explicit revocation is not needed for the VDS protocol, which updates the public key on every update thereby rendering old versions invalid.

6.3 Active Network Attacker

The active network attacker can modify or replace data packets containing documents or signatures. However, any modification or replacement would either result in an invalid signature since the document and its corresponding signature would no longer match, or result in a document signed with the untrusted key of the attacker. The attacker cannot access the secret key that was used by the content author to sign the original data and hence cannot re-compute the signature. The attacker can also try to substitute the response with a valid response of another or older document. Replacement with another document is prevented by the embedded absolute URI as part of the signature chain, which reveals the substitution. The older document is prevented by the previously described freshness properties. Since HTTP is per se stateless, session cookies are often deployed to transfer state information. This way, the same URI can transfer different Web resources. WebTrust allows to uniquely identify these different documents, since the session information is part of the signed HTTP header.

6.4 Active Attacker against CDN and Web Server

Our solution successfully protects against the CDN Attacker. The CDN stores solely documents that are already WebTrust protected. If an attacker exploits a known vulnerability in the CDN server, documents can be replaced. However, the attacker cannot forge valid signatures for this malicious content since there

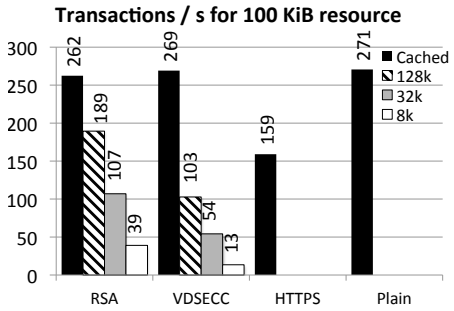

Figure 7. Average transactions per second under maximum load

is no possibility to access the required secret key. Hence, our solution preserves the authenticity and integrity of documents. Since we assume that static and dynamic content signing takes place at the content generator where an attacker cannot gain access, content can only be manipulated after it has left the content generation server. In this stage, it is already digitally signed and can no longer be manipulated without detection.

7 Experimental Evaluation

In the following, we provide the experimental evaluation of our prototypical WebTrust implementation. The evaluation encompasses the computational overhead at the client and the content generator as well as the network overhead. However, the measured overhead does not include any processing of or communication with the PKI, which may slightly skew the measured performance. Moreover, we discuss usability issues of the current implementation.

7.1 Performance Evaluation

We conducted a comprehensive performance evaluation to measure the runtime and network overhead induced by WebTrust. The server side evaluation was performed with our patched Apache Tomcat version running on a Dell Optiplex 9010 Workstation equipped with an Intel Core i7 CPU and 32 GB of Ram. The client side evaluation was performed on an identical machine with 16 GB of Ram instead. We chose the security parameters of the cryptographic schemes according to the latest NIST recommendations [24], i.e. 2048 bit for RSA. To achieve a comparable level of security, we chose the elliptic curve `P-224` for the chameleon hashes inside the CAT.

The performance and network overhead depends on the total amount of signatures that are sent over the network, i.e. the ratio between transmitted bytes of data and transmitted bytes used for signatures. The smaller the size of each

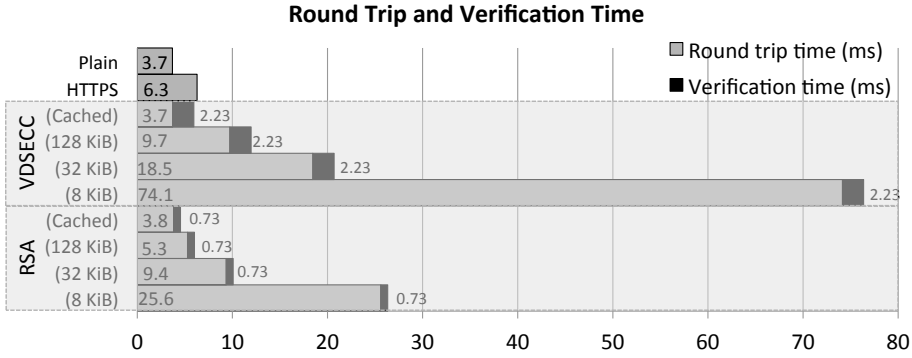

Figure 8. Round trip times for a 100 KB document

chunk, the more signatures are need for the same amount of data. In our prototype, we evaluated different chunk sizes, namely 8 KB, 32 KB and 128 KB. If we consider a video stream of moderate standard definition quality with 2 Mbit/s data rate, then 128 KB chunks would correspond to one second of video – which seems to be an acceptable frequency for content verification. The experiments were conducted using the Siege [36] benchmark tool that downloaded a 100 KB file 10,000 times while simulating 100 concurrent users accessing the server. The server and the client were both connected to the Internet via a 1 Gbit/s uplink and they are 11 hops apart. We measured the maximum transactions that the server was capable of delivering without WebTrust, with WebTrust RSA and VDSECC and over HTTPS (see in Figure 7).

This number of transactions is limited by the computational burden on the server side. The client-side verification adds an additional delay for verifying each chunk. This delay is $121\mu s$ for one RSA signature verification and $371\mu s$ for one CAT node verification. The resulting round trip times are depicted in Figure 8. Our results show that in the case of static content, the cached versions have a negligible overhead compared to plain HTTP connections. Without server-side caching, i.e. the server has to calculate the corresponding RSA signatures or CAT trees for each chunk, the computational load on the server side increases. However, RSA signature chaining still outperforms HTTPS with RSA 2048 bit Diffie-Hellman key exchange and AES-256-CBC encryption. Even though the VDSECC-based authentication without caching provides less throughput than HTTPS, VDSECC is the only primitive whose revocation mechanism does not need to keep a list for each revoked document (see section 4).

Network Overhead. WebTrust introduces a small overhead in size for every signature that is transmitted from the Web server to the client. The overhead for a single signature depends on the signature scheme and its security parameter. The signature size of RSA 2048 bit is 344 bytes. The size of one VDSECC data structure is 167 bytes. These sizes resemble a space overhead of 4% (RSA) and

2% (VDSECC) in the worst case of very small 8 KiB sized chunks. For more realistic sizes of 128 KB chunks, the overhead is a mere 0.3% for RSA and 0.1% for VDSECC, respectively.

7.2 Usability

The prototypical implementation seamlessly integrates into HTTP's chunked transfer encoding, which provides full backward compatibility. At the client-side, the prototype is integrated into Chromium and uses in addition an extension for providing the individual verifiability.

Discussion. If the distinctive features of the CAT are not required and bandwidth considerations are less important than the computational overhead RSA is the primitive of choice. Otherwise CAT provides the full set of functionality at a speed that is still reasonable for today's Internet connections. Our round-trip time measurement indicates that delay introduced by the network transmission still dominates the computation times the client and the server. The CAT-based solution leaves still room for performance optimizations. Depending on the use-case one could pre-calculate several keys to further reduce the computational load [5], especially on the client side. Moreover, we did not use multi-threading for verifying chunks simultaneously. Depending on the scenario one could also reduce both the computational and the network overhead can be tweaked by changing the verification ratio via the chunk size.

8 Conclusion

Verifying the integrity and authenticity of dynamic Web content and real-time Web streams on-the-fly is infeasible with existing solutions. Motivated by this lack of solutions, we developed WebTrust, the first comprehensive solution to provide integrity in all major Web scenarios. WebTrust allows to verify integrity and authenticity of static, dynamic, and streamed Web content and integrates seamlessly into the existing Web infrastructure. Our performance results demonstrate both its practicality and efficiency, even in the mobile setting. The results of our evaluation show that there is not one primitive for all scenarios that clearly outperforms all others. Which technique and which cryptographic primitive to use highly depends on the task since no primitive provides all security features, a small overhead in size, enables caching, and provides a high performance both on the client and the server side.

Acknowledgement. We thank the anonymous reviewers for their comments and Oliver Schranz for his assistance with the implementation. This work was supported by the German Ministry for Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA).

References

1. Ateniese, G., de Medeiros, B.: On the Key Exposure Problem in Chameleon Hashes. In: Proc. of the 4th International Conference on Security in Communication Networks (SCN 2004). LNCS, vol. 3352, pp. 165–179. Springer (2004)
2. Bayardo, R.J., Sorensen, J.S.: Merkle tree authentication of HTTP responses. In: Proc. of the 14th International Conference on World Wide Web (WWW 2005). pp. 1182–1183. ACM (2005)
3. Bellare, M., Rogaway, P.: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: Proc. of the 1st ACM Conference on Computer and Communication Security (CCS 1993). pp. 62–73. ACM (1993)
4. bouncycastle.org: The Legion of the Bouncy Castle. Online at `http://www.bouncycastle.org/` (2013)
5. Catalano, D., Fiore, D., Gennaro, R.: Certificateless onion routing. In: Proc. of the 16th ACM Conference on Computer and Communication Security (CCS 2009). pp. 151–160. ACM (2009)
6. Choi, T., Gouda, M.G.: HTTPI: An HTTP with Integrity. In: Proc. of the 20th International Conference on Computer Communications and Networks (ICCCN 2011). pp. 1–6. IEEE Computer Society (2011)
7. The Chromium Projects. Online at `http://www.chromium.org/` (2014)
8. Devanbu, P., Gertz, M., Kwong, A., Martel, C., Nuckolls, G., Stubblebine, S.G.: Flexible Authentication Of XML documents. In: Proc. of the 8th ACM Conference on Computer and Communication Security (CCS 2001). pp. 136–145. ACM (2001)
9. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. Online at `http://tools.ietf.org/html/rfc2616` (1999)
10. Fox, A., Brewer, E.A.: Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. In: Proc. of the 5th International Conference on World Wide Web (WWW 1996). pp. 1445–1456. Elsevier (1996)
11. Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Stewart, L.: RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication. Online at `http://tools.ietf.org/html/rfc2617` (1999)
12. Gaspard, C., Goldberg, S., Itani, W., Bertino, E., Nita-Rotaru, C.: Sine: Cache-friendly integrity for the web. In: Proc. of the 5th IEEE Workshop on Secure Network Protocols (NPSec 2009). pp. 7–12. IEEE Computer Society (2009)
13. Gennaro, R., Rohatgi, P.: How to sign digital streams. In: Proc. of Advances in Cryptology (CRYPTO 97). LNCS, vol. 1294, pp. 180–197. Springer (1997)
14. Gionta, J., Ning, P., Zhang, X.: iHTTP: Efficient Authentication of Non-confidential HTTP Traffic. In: Proc. of the 10th International Conference on Applied Cryptography and Network Security (ACNS 2012). pp. 381–399. Springer (2012)
15. Hohenberger, S., Waters, B.: Realizing Hash-and-Sign Signatures under Standard Assumptions. In: Proc. of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 09). LNCS, vol. 5479, pp. 333–350. Springer (2009)
16. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)
17. Krawczyk, H., Rabin, T.: Chameleon Signatures. In: Proc. of the 7th Annual Network and Distributed System Security Symposium (NDSS 2000). The Internet Society (2000)

18. Lesniewski-Laas, C., Kaashoek, M.F.: SSL Splitting: Securely Serving Data from Untrusted Caches. In: Proc. of the 12th Usenix Security Symposium. pp. 187–199. Usenix Association (2003)
19. Lesniewski-Laas, C., Kaashoek, M.F.: SSL splitting: Securely serving data from untrusted caches. Computer Networks 48(5), 763–779 (2005)
20. Lin, C.Y., Chang, S.F.: Generating robust digital signature for image/video authentication. In: Proc. of the 1st Workshop on Multimedia and Security at ACM Multimedia 1998. vol. 98, pp. 94–108. ACM (1998)
21. Merkle, R.C.: Method of Providing Digital Signatures (US Patent: US4309569A) (1979)
22. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Proc. of Advances in Cryptology (CRYPTO 85). LNCS, vol. 218, pp. 417–426. Springer (1985)
23. Moyer, T., Butler, K.R.B., Schiffman, J., McDaniel, P., Jaeger, T.: Scalable Web Content Attestation. IEEE Transactions on Computers 61(5), 686–699 (2012)
24. NIST: Recommendation for Key Management. Special Publication 800-57 Part 1 Rev. 3 (2012)
25. OpenSSL. Online at `http://www.openssl.org/` (2014)
26. Oracle: Java Cryptography Architecture – Oracle Providers Documentation. Online at `http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html` (2013)
27. Pannetrat, A., Molva, R.: Efficient Multicast Packet Authentication. In: Proc. of the 10th Annual Network and Distributed System Security Symposium (NDSS 2003). The Internet Society (2003)
28. Perrig, A., Canetti, R., Tygar, D., Song, D.: Efficient authentication and signing of multicast streams over lossy channels. In: Proc. of the 2000 IEEE Symposium on Security and Privacy (Oakland 2000). pp. 56–73. IEEE Computer Society (2000)
29. Ray, I., Kim, E.: Collective Signature for Efficient Authentication of XML Documents. In: Proc. of the IFIP TC-11 19th International Information Security Conference (SEC 2004). pp. 411–424. Springer (2004)
30. Reis, C., Gribble, S.D., Kohno, T., Weaver, N.C.: Detecting In-Flight Page Changes with Web Tripwires. In: Proc. of the 5th Usenix Symposium on Networked Systems Design and Implementation (NSDI 2008). pp. 31–44. Usenix Association (2008)
31. Rescorla, E.: RFC 2818 - HTTP Over TLS. Online at `http://tools.ietf.org/html/rfc2818` (2000)
32. Rescorla, E., Schiffman, A.: RFC 2660 - The Secure HyperText Transfer Protocol. Online at `http://tools.ietf.org/html/rfc2660` (1999)
33. Rivest, R.L., Shamir, A., Adleman, L.M.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM (CACM) 21(2), 120–126 (1978)
34. Schröder, D., Schröder, H.: Verifiable data streaming. In: Proc. of the 19th ACM Conference on Computer and Communication Security (CCS 2012). pp. 953–964. ACM (2012)
35. Shamir, A., Tauman, Y.: Improved Online/Offline Signature Schemes. In: Proc. of Advances in Cryptology (CRYPTO 01). LNCS, vol. 2139, pp. 355–367. Springer (2001)
36. Siege Home. Online at `http://www.joedog.org/siege-home/` (2014)
37. Singh, K., Wang, H.J., Moshchuk, A., Jackson, C., Lee, W.: Practical End-to-End Web Content Integrity. In: Proc. of the 21st International Conference on World Wide Web (WWW 2012). pp. 659–668. ACM (2012)