Vol.2022-DPS-190 No.16
Vol.2022-CSEC-96 No.16
2022/3/10

Mutual Declaration Mechanism of Multi-provider Relationship for Trusted Web Services

Wataru Ohgai[1,a)] Takao Kondo[,b)] Korry Luke[3,c)] Satoshi Kai[4,d)]
Keisuke Uehara[5,e)] Satoru Tezuka[5,f)]

Abstract: The TLS security model enables the identification and secrecy of the host-to-host communication channel on the Web; however, TLS cannot guarantee the relationship between service providers. This paper proposes a lightweight self-managed mutual declaration mechanism, M2DMRT, in which service providers mutually sign their TLS public keys and publish them in DNSSEC-protected DNS zones. With M2DMRT, service providers can mutually declare their relationships with each other, and end users can easily trust the relationships by verifying the signatures. Further, this paper implemented a server-side proof of concept. After evaluating its basic performance and feasibility from an Internet architecture perspective, this paper found this mechanism can realize more trustable Web security architecture by providing a sufficiently performant way to declare and verify relationships between service providers without significantly impacting the current Internet environment.

1. Introduction

The Web security model is based on host-to-host encryption and integrity provided by Transport Layer Security (TLS) [1]. TLS ensures the integrity of the host based on verification of the host's public key certificate by Web PKI and encrypts communication by generating a shared key with forward secrecy using Diffie-Hellman Ephemeral (DHE) [2] / Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) [3] key exchange.

Web service providers deal with an increasing number of website functions, end users, and overall traffic volume, generally by building complex backend infrastructure. Therefore, many Web services are built by rerouting traffic (e.g., HTTP redirection) from one service provider to another, through the use of third parties such as Platform as a Service (PaaS) or Content Delivery Networks (CDN). However, in order to define single host in TLS host-to-host security model, it requires separate public key certificates, public keys, and private keys per host, which amplifies operational costs in Web service provider. To reduce TLS operational costs, such services often share their public key certificates across different domains by adding Subject Alternative Names (SAN) [4] or defining content rerouting policy using Cross-Origin Resource Sharing (CORS) [5].

However, this security model does not provide any assurance of integrity when rerouting service traffic. TLS can ensure communications integrity with a single host but cannot ensure the integrity of an entire service when requests are redirected across several distinct service providers. Thus the end users can be redirected to the malicious hosts or communications between the end users' client and the service providers' hosts can be monitored because the end users are unaware of and have no way to trust the integrity of entire service.

HTTP Strict Transport Security (HSTS) [6] allows site operators to instruct clients to always use TLS when connecting to a specific domain, which can mitigate some of the threats associated with machine-in-the-middle content modification, DNS spoofing, and ARP spoofing, which represent potential attacks on the confidentiality of data sent to a specific site.

Since the decision to use HSTS lies with the service provider, TLS downgrade attacks [7, 8] are still possible in non-HSTS compliant services, or in services which send HSTS header to the client when the client try to connect via HTTP without TLS at first. To avoid disabling HSTS when the client try to connect via HTTP without TLS at first, major Web browsers like Mozilla Firefox [9] and Google Chrome [10] are migrating to "HTTPS by default" policy, which does not try to use HTTP without TLS. Also, HSTS preload list [11] is hardcorded into Google Chrome and other

[1] Faculty of Policy Management, Keio University
[2] Computer Security Incident Response Team, Keio University
[3] Information Technology Center, Keio University
[4] Keio Research Institute at SFC, Keio University
[5] Faculty of Environment and Information Studies, Keio University
[a)] alt@sfc.wide.ad.jp
[b)] latte@itc.keio.ac.jp
[c)] koluke@sfc.wide.ad.jp
[d)] skai@sfc.keio.ac.jp
[e)] kei@wide.ad.jp
[f)] tezuka@sfc.keio.ac.jp

major Web browsers, though it is not standardised.

Even in services which do support HSTS, existing research [12] has shown that attackers can disable HTTPS enforcement in certain conditions, such as when several hosts share the same TLS certificate [13]. This research focuses on the threat model around rerouting service traffic across service providers located in different domains.

To address this threat, this paper proposes *M2DMRT* (*Mechanism of Mutual Declaration of Multi-provider Relationship for Trusted Web Services*), a mechanism by which several service providers forming one service can declare their relationship mutually. M2DMRT assumes that end users can already verify each hosts' communications integrity using TLS and the existing Web PKI.

In the proposed mechanism, the reroute origin service provider uses their TLS private key to sign the redirect destination service provider's TLS public key. Similarly, the reroute destination service provider uses their TLS private key to sign the redirect origin service provider's TLS public key. After both signatures are generated, the signature and hostname corresponding to each TLS public key's certificate is registered in service provider's DNSSEC-protected DNS zone authoritative server. The end users can verify the relationship by querying DNS and verifying DNSSEC integrity, the mutual signature provided by M2DMRT, and the host integrity.

2. Requirements Definition

2.1 Definition of the Threat Model

As described in Sec. 1, this paper defines a threat model for Web services in which either malicious attacks or operational failures [14] cause traffic to be rerouted to or from a malicious service provider in a manner not intended by the genuine service provider.

Sec. 1 describes the current situation of backend infrastructure of Web service providers, many of which use multiple services, domain names, hosts to construct a single Web service. This paper defines "service integrity" as the integrity of a single Web service from the end user perspective, which may consist of several logically distinct Web services, domain names, or hosts.

Currently, there are no existing mechanisms to assure service integrity on the Internet. This is because the security model of Web has been discussed based on host integrity assurance by TLS and Web PKI. The lack of Service Integrity security model, which means only depends on the host itself but the entire Web service, is causing threats described in Sec. 1.

A single Web service can be consisted by a number of host chained both vertically and horizontally.

In a vertically-chained hosts environment, a single service provider may have multiple hosts related with other service providers' hosts. Such cases are assumed to exist widely in cloud or CDN environments.

In a horizontally-chained hosts environment, many service providers may have relationship when constructing a single service integrity. Figure 1 shows an example environment of single online shopping service constructed by three services, (i) EC site service provider that actually sells and ships products physically, (ii) payment processing site service provider that collects credit card information from users, and (iii) 3-D Secure [15] site service provider that provides authentication for online credit card use.

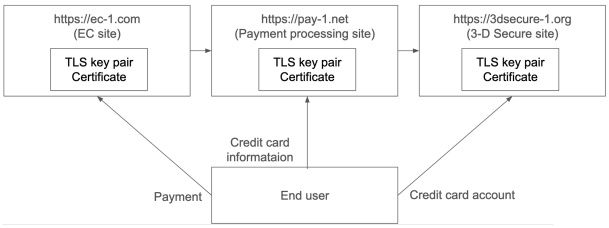

Fig. 1 Example environment of horizontally-chained hosts.

Considering the popularity and increasing adoption of cloud services like PaaS, a single service provider may have a number of different relationships with other distinct service providers. In such cases, the PaaS provider should bear the responsibility of keeping track of its own relationship relative to other related service providers.

This paper addresses threat models around service integrity assurance which have not been addressed previously in the security model of host integrity assurance. Specifically, these include attacks or vulnerabilities of the nature of host integrity, which can be affected by HTTPS hijacking attacks or subdomain takeover attacks [16, 17] in CDN or cloud environment. These threat models can be addressed by combination of host integrity assurance and service integrity assurance to identify and verify single hosts in the relationship and assuring the relationship among multiple hosts.

2.2 Requirements

2.2.1 Requirements against the Threat Model

To address this threat model, this research defines three requirements against the threat model that must be met (Req. #1. – Req. #3.), along with two additional supporting requirements that may make this proposal more architecturally feasible in the real world (Req. #4, 5).

Req. #1. Mutual and Verifiable Declaration of Service Relationship In order to prevent rerouting both to and from malicious service providers, this declaration must be mutual. In addition, to enable every end users to verify this declaration using trust anchor trusted by anyone, this declaration must be verifiable.

Req. #2. Self-manageable Declaration of Service Relationship Relationship between service providers is based on their actual business contract. This declaration should not be modifiable unless both service providers have intentions to do.

Req. #3. Minimum Disclosure of Each Party's Components From a business and cyber security perspective, the disclosure of endpoints configuration in each service provider is sensitive. Thus, the disclosure should be as min-

imal as possible.

Req. #4. Localization of Transaction of Modification This mechanism should be scalable for use in global platforms, thus modification transactions should be localized.

Req. #5. Localization/Minimization of Failure Points (Independent from Central Authority) This mechanism should have availability, thus this mechanism should not rely on a centralized topology or have specific single points of failure.

2.2.2 Requirements for Feasibility

From a real-world operations perspective, this paper also requires the mechanism to be work efficiently in a manner compatible with existing current Internet architecture.

Req. #6. Adoption to the Vertically-chained Environment The mechanism should be able to hold the complex state of relationship in vertically-chained environment to assure single service integrity in practical use.

Req. #7. Adoption Potential in Horizontally-chained Environments The mechanism should be able to hold the complex state of relationship in horizontally-chained environment to assure single service integrity in practical use.

Req. #8. Minimal Processing Time of Modification Operating new mechanism generally requires additional time consumption. In order to reduce down time of the services, the processing time of modification of declaration must be minimized.

Req. #9. Mitigation of Load of PaaS Service Provider The mechanism should be designed to mitigate the load on relatively large service providers described in Sec. 2.1.

3. Related Works

3.1 HTTPS-based Mechanisms

Certificate Transparency (CT) [18] is a mechanism that allows network administrators and Web site operators to monitor a set of public logs for misissued certificates for a specific domain. In its current implementation, end users are not able to determine whether a certificate has been issued mistakenly or belongs to a compromised site independently, meaning end users may still connect to a malicious or compromised Web site. Using CT, certificates issued from a Certification Authority (CA) are registered to a CT log server run by a third party, which returns a signed timestamp known as a Signed Certificate Timestamp (SCT). Verification of the certificate using CT is done by checking if the CT log server has an entry corresponding to a specific certificate based on the SCT. If this verification fails, the client can assume the certificate has not been logged publicly and may be suspect since other members of the Web PKI ecosystem have not been able to confirm its policy compliance. As a result, certificates are in practice forced to be publicly disclosed in order to be trusted by browsers that implement CT.

Cross-Origin Resource Sharing (CORS) [5] is a mechanism that allows clients to send HTTP requests to another server when the same-origin policy [19] is defined in the Web application. Same Origin Policy plays an important role to protect from Web security threats such as cross-site scripting (XSS) or cross-site request forgery (CSRF) attacks. When CORS is applied, the *Origin* header is used to indicate the requesting domain to the server that allows resource sharing. The Web server which received HTTP request with *Origin* field can read the provided header and allow access to clients of the original Web server. When the original Web server is successfully authenticated, *Access-Control-Allow-Origin* field is added to the HTTP response header so that the client can identify the authenticated origin name.

Subresource Integrity (SRI) [20] addresses the threat of a subresource (such as a JavaScript library or stylesheet) being modified when hosted by a third party, such as a CDN. When a Web service includes subresources hosted by a third party referred using a *script* or *link* tag, the original Web server provides the expected hash of the subresource in an *integrity* attribute. The client can calculate the hash of the downloaded subresource and verify whether it matches the original site's intended hash; resources that do not match can be rejected easily, limiting the damage caused by a third-party CDN compromise.

3.2 Application of Distributed Repository

Some researchers are trying to reconstruct DNS [21, 22] using Distributed Hash Table (DHT) [23, 24] or Blockchain [25].

Handshake [26] is a distributed naming protocol using Blockchain. Peers in the Handshake network can create or verify transactions that represent management changes to top-level domain names. Also, Handshake Blockchain proves the manager of the domain name, thus Handshake can also act as a CA of sorts for use with PKI. The Handshake model encourages distribution by providing economic incentive for transactions and proof of administrative rights for domains to Handshake users.

3.3 DNSSEC and DANE TLSA

DNSSEC [27] serves two main functions: (i) assurance of DNS zone delegation in the global name space by using root DNS zone's Key Signing Key (KSK) as trust anchor, and (ii) assurance of response contents from the authoritative servers and its authenticity.

DNSSEC also provides denial of existence in response to the query for non-existing domain name to assure integrity of entire DNS zone. This is realized by using *NSEC* resource record and its signature by ZSK public key, which contains the alphabetically next domain name of queried domain name (*QNAME*). *NSEC3* [28] resource record is also used for hashed denial of existence to avoid listing up all of the domain name in a DNS zone.

DANE TLSA [29] is a protocol which allows a DNS zone administrator to designate a particular host's TLS public key certificate or authorized issuing CA using DNSSEC. The

Table 1 Comparison of requirements against the threat model and related works.

	CT [18]	CORS [5]/ SRI [20]	DHT [24]/ Handshake [26]	DANE TLSA [29]
Req. #1.	No	No	No	No
Req. #2.	No	No	No	No
Req. #3.	No	No	No	Yes
Req. #4.	Partial	Yes	No	Yes
Req. #5.	No	Yes	Yes	Yes

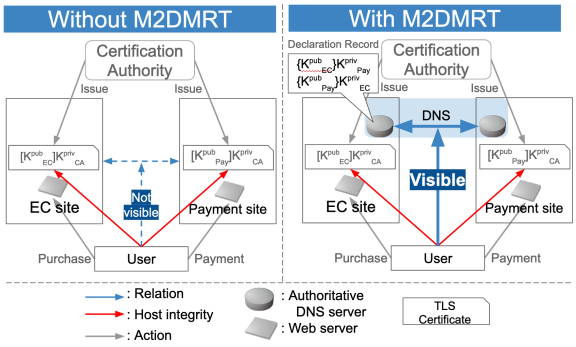

Fig. 2 Assumed environment of M2DMRT.

DNS zone administrator can declare how clients should verify certificates and can even choose to instruct clients to ignore Web PKI trust chain and rely on the DNSSEC-provided chain only. The use of DANE TLSA is limited to specific hosts in a given DNS zone, thus it is impractical for designating several hosts at once and hosts whose DNS resource records resolve outside of a specific DNS zone.

3.4 Comparison of Related Works and Requirements against Threat Model

Table 1 shows the comparison of each related works and requirements defined in Sec. 2. This research evaluated localization of transaction for Web PKI and CT as "partial" since transactions are localized between the Web server and CA but span multiple domains. None of the related works surveyed satisfies all of Req. #1. – 3., thus even using combinations of existing technologies simultaneously cannot satisfy the goal of this research.

4. Proposal: M2DMRT

4.1 Assumed Environment

This paper assumes a Web service that consists of two service providers that reroute service traffic using HTTP redirects. One is a hypothetical e-commerce (EC) site that operates an online store and redirects end users to the second site during the checkout process, and the other is a payment processor site that processes credit card payment transactions for end users redirected from the EC site.

The left half of Figure 2 shows the current Web service situation, protected only by the host integrity assurances of TLS. An end user wants to buy some product from the EC site is redirected to a payment processor and asked to enter credit card information. The end user's client is communicating to each Web service providers' Web servers using TLS, thus the client can use TLS certificates provided by each server to verify each individual server's integrity using the Web PKI, with Web PKI CA certificates serving as a trust anchor.

However, the end user cannot verify the relationship between the EC site and payment site easily; thus the end user may be reluctant to provide credit card information to the payment site, even though the payment site may be providing services on the EC site's behalf. Furthermore, the end user may have been redirected to a malicious server by HTTPS hijacking attack or subdomain takeover attack on CDN or cloud environment.

The right half of Figure 2 shows an environment which uses both TLS and service integrity assurance provided by M2DMRT. In M2DMRT, service providers sign the other service provider's TLS public key using own TLS private key each other. The signatures are published in the service provider's DNSSEC-protected DNS authoritative server as DNS resource records (*declaration records*). DNS and DNSSEC provides self-manageability of declaration records by service providers, decentralized structure, and assurance that the any generation, modification or deletion is based on the service providers' intention. Every TLS keypair used by the service providers are signed mutually, with each signature is published as a distinguished declaration record; thus the service providers are not required to publish all of their endpoints as a single list. End users can easily verify relationship between service providers by using M2DMRT.

4.2 Architectural View of M2DMRT

Figure 3 shows the architectural view of M2DMRT. In this figure, this paper assumes that an application service provider named *ec-1.com.* has a business contract with a payment processor provider named *pay-1.net.*. Likewise, the other application service provider named *ec-2.com.* has business contract with the other payment processor provider named *pay-2.net.*. This paper also assumes that DNSSEC is enabled for the DNS zones of . (root), *com.*, *ec-1.com.* and *ec-2.com.*.

Focusing on the relationship between *ec-1.com.* and *pay-1.net.*, the declaration records shown on the right half of Figure 2 are registered in form of DNS TXT type resource records in the DNS zone of *ec-1.com.* and signed by the ZSK of *ec-1.com.*. The domain name of declaration records consists of the format "*destination_hostname.*_m2dmrt.*origin_hostname.*" The data part of a declaration record consists of the hostname of the signer TLS private key, the hostname of the target TLS public key, and the signature itself. Since the declaration is mutual, two declaration records are generated for each relationship. In order to prevent DNS zone enumeration attacks (also known as "DNS zone walking"), NSEC3 should be used.

4.3 Registration Sequence

Figure 4 shows the authentication and key exchange sequence between MRDAs belonging to the EC site and payment site. Both service providers register their TLS public

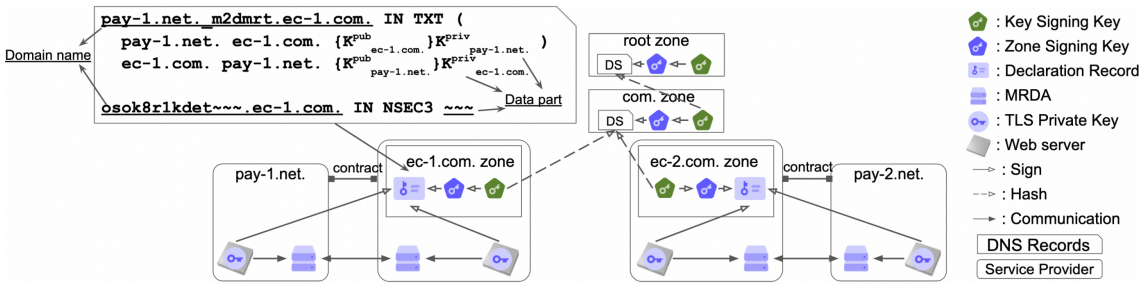

Fig. 3 Architectural view of M2DMRT.

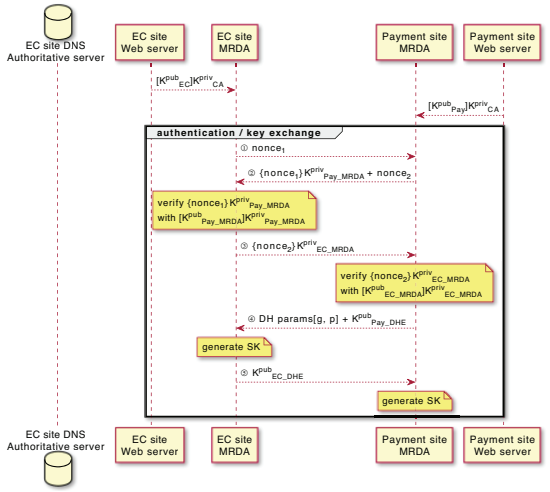

Fig. 4 Authentication / key exchange procedures in the registration.

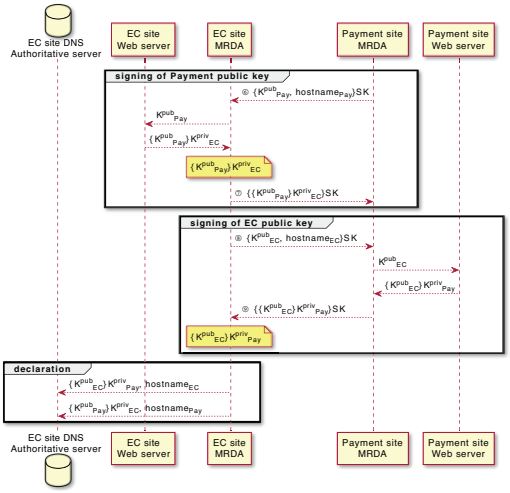

Fig. 5 Signing of TLS public keys / declaration procedures in the registration.

keys to their respective MRDAs, create RSA keypairs, and exchange the RSA public keys securely beforehand.

When beginning registration for a new declaration, both MRDAs authenticate each other in a CHAP-like process and perform DHE key exchange. In Figure 4–(1) – Figure 4–(3), MRDAs send nonce ($nonce_1$, $nonce_2$) and authenticate each other by verifying a nonce signed by their counterpart's private key against their counterpart's public key certificate. In Figure 4–(4), Figure 4–(5), MRDAs perform DHE key exchange to derive a shared key (SK) and open an AES-protected communication channel.

Figure 5 shows the sequence of signing of TLS public key by MRDA and publishing of declaration records in DNS authoritative server.

Next, MRDAs sign the target's TLS public key with their side's TLS private key. Figure 5–(6), Figure 5–(7) shows signing steps of payment site TLS public key, and Figure 5–(8), Figure 5–(9) shows signing steps of the EC site's TLS public key. M2DMRT is still feasible if one service provider has several Web servers with multiple TLS keypairs. MRDAs can identify which host's TLS public key they are signing by sending the hostname of the Web server using that TLS public key when MRDAs request a signature generation (Figure 5–(6), Figure 5–(8)). The TLS private keys used in signature generation are not managed by MRDAs but Web servers, thus MRDAs need to make requests to signing agents running inside the Web server infrastructure. Generated signatures are returned to the original MRDA to verify that TLS public key is successfully signed. If this verification fails, the MRDA should request a new signature.

Finally, signatures are published as declaration records by sending a dynamic update [30] query from the MRDA to a DNS authoritative server using TSIG [31]. The number of resource records corresponds directly to the number of pairs of signing TLS private keys and signed TLS public keys.

4.4 Verification Sequence

Figure 6 shows the verification sequence of steps of M2DMRT when the user is redirected from *ec-1.com.* to *pay-1.net.*.

At first, the user's device connects to the Web server of *ec-1.com.* via HTTPS. At this time, the M2DMRT client implemented in the user's device obtains the public key certificate of *ec-1.com.*, which includes the actual public key. When the user's device issues the HTTP query that results in redirection to *pay-1.net.*, the Web server of *ec-1.com.* returns an HTTP 3xx response.

Second, before accessing the Web server of *pay-1.net.*, the M2DMRT client performs signature verification.

To obtain the M2DMRT signatures, the M2DMRT client resolves the TXT resource record of *pay-1.net._m2dmrt.ec-1.com.* using DNSSEC. In response to the query, the DNS authoritative server of *ec-1.com.* should return a set of declaration records which is signed by DNSSEC ZSK of *ec-1.com.* with a NOERROR RCODE. If the M2DMRT client receives an NXDOMAIN response with NSEC3 hashed de-

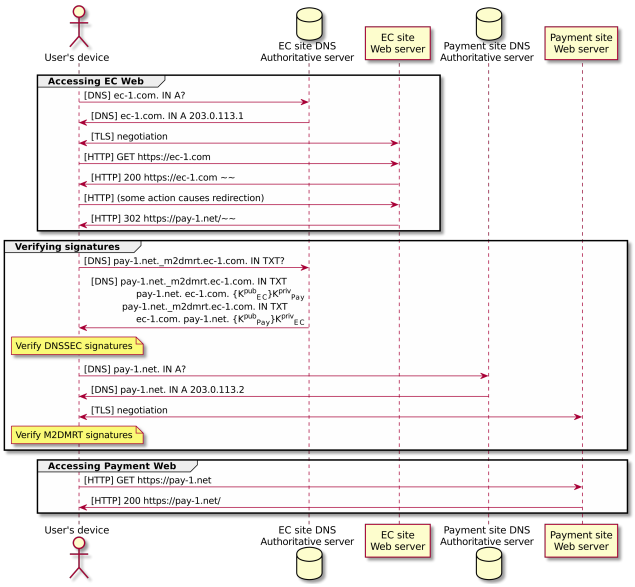

Fig. 6 Verification sequence.

nial of existence, it can assume that the service providers do not support M2DMRT.

After the DNSSEC verification, the M2DMRT client attempts to connect to the *pay-1.net.* Web server with TLS. During the TLS negotiation, the M2DMRT client can obtain the public key certificate of *pay-1.net.* and the corresponding public key itself. Using both service providers' public keys and declaration records, the M2DMRT client performs signature verification on the declaration. If the verification fails, the M2DMRT client can reattempt the verification sequence, or instruct the browser to terminate communication with *pay-1.net.*.

After completing both verification of DNSSEC and M2DMRT signatures successfully, the browser can continue to connect to the *pay-1.net.* Web server via HTTPS with confidence the connection is trustworthy.

5. Experiment and Evaluation

5.1 Experiment using PoC

This paper performed 2 experiments to measure in-node processing time of our proof of concept implementation.

In the first experiment, this paper used *time* library to measure the cryptographic processing time of simulated declaration record registration by calculating the difference in timestamps. This experiment used Apple macOS Big Sur 11.4 on M1 CPU (3.2 GHz, 8 cores) with 16 GB RAM.

The second experiment measured the entire processing time of the simulated modification of declaration record using the shell script. This experiment used Knot DNS 7.2.8 running on Ubuntu 20.04.3 LTS on VMware ESXi 7.0 with 4 vCPUs, 16 GB RAM.

5.2 Quantitative Evaluation against PoC

Figure 7 and Figure 8 show the evaluated in-node processing time ranges of cryptographic processes.

Table 2 and Table 3 show the average time of 100 trial runs for each time range. Measurement was performed in an environment assuming two service providers which manage one Web server and one TLS keypair each.

Table 2 Average time of each range regarding authentication and key exchange.

(1)	(2)	(3)	(4)	(5)
11.7 ms	0.616 ms	11.2 ms	0.409 ms	49.0 ms
(6)	(7)	(8)	-	-
6.87 ms	0.227 ms	0.282 ms	-	-

Table 3 Average time of each range regarding TLS public key signing.

(9)	(10)	(11)	(12)	(13)
1.43 ms	1.40 ms	11.0 ms	0.366 ms	0.379 ms
(14)	(15)	(16)	(17)	(18)
0.20 ms	0.183 ms	10.9 ms	0.176 ms	0.201 ms
(19)	-	-	-	-
0.393 ms	-	-	-	-

In Table 2, Table 2–(1) and Table 2–(3) express time for signing *nonce* by MRDA's private key, Table 2–(2) and Table 2–(4) express time for verification of the signature generated in Table 2–(1) and Table 2–(3) by MRDA's public key, Table 2–(5) and Table 2–(6) express time for calculating DHE public key (DHK) of payment site's MRDA, Table 2–(7) and Table 2–(8) express time for calculating DHE shared key (SK) for AES encryption and decryption. Verification of signatures and calculation of SK are finished faster enough compared to signing of *nonce* or calculation of DHK.

For evaluation of authentication and key exchange, generation of signature against nonce (Table 2–(1), (3)) and calculation of DHE public key (Table 2–(5), (6)) take longer time; however, verification of nonce signature (Table 2–(2), (4)) and calculation of DHE shared key (Table 2–(7), (8)) is processed faster.

In Table 3, Table 3–(9), Table 3–(14), and Table 3–(17) express time for AES encryption of the message by SK, Table 3–(10), Table 3–(12), Table 3–(15) and Table 3–(18) express time for AES decryption of the message by SK, Table 3–(11) and Table 3–(16) express time for signing TLS public key by TLS private key of the opponent, Table 3–(13) and Table 3–(19) express time for verification of signature generated in Table 3–(11) and Table 3–(16).

For evaluation of generating signature of TLS public key (Table 3–(11), (16)), which is the core of this mechanism, can be processed in about 10 msec.

This paper measured average processing time of registration. In the experiment, this paper used *exp.jj1lfc.dev.* DNS zone for both simulated EC site and payment site to utilize DNSSEC environment of the global DNS.

The average time took for registration was 676.6 msec.

5.3 Qualitative Evaluation against Threat Model Requirements

This section evaluates how M2DMRT satisfies three requirements against threat model defined in Sec. 2.2.1 (Req. #1. – 3.). M2DMRT's fulfillment of system requirements (Req. #4., 5.) by using DNS and DNSSEC is as

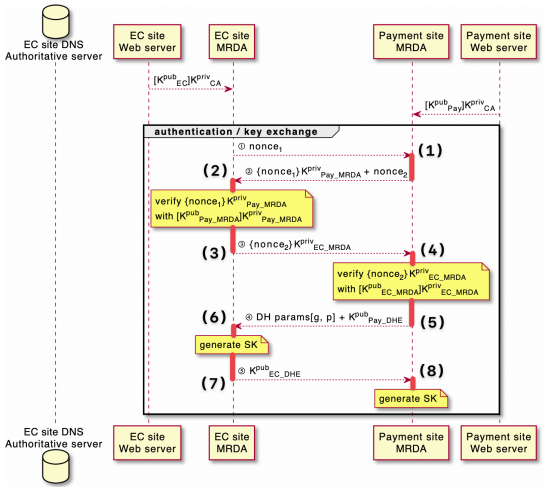

Fig. 7 Evaluated time range of authentication / key exchange procedures.

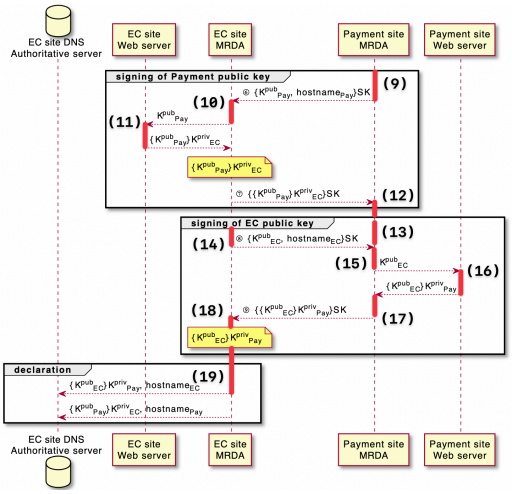

Fig. 8 Evaluated time range of signing of TLS public keys / declaration procedures.

described previously in Sec. 3.3.

Req. #1.: Declaration records are mutually signed by both reroute origin and destination service providers. Registration, modification, and deletion of declaration records can only be done through mutual authentication of both MRDAs using MRDAs' asymmetric keypairs. Thus, end users are protected against redirection both to and from malicious service providers. Since declaration records are published in DNSSEC-protected DNS zones, end users can access and verify declaration records easily.

Req. #2.: Signatures require both service providers' TLS private keys to be generated. Since these keys are held directly by the service providers themselves, providers who have agreed mutually can manage their declaration records on their own without reliance upon third-party external infrastructure.

Req. #3.: The redirect origin and destination hostname are both necessary to query the declaration records successfully. Acquiring these hostnames through DNS zone-walking is prevented by using NSEC3 hashed denial of existence. As a result, it is relatively difficult for an attacker to list all endpoints involved in a transaction using M2DMRT.

Req. #4.: Modification transactions is mainly performed by MRDAs in each service provider. As described in Sec. 3.3, after signature declaration, the declaration records are published through DNS authoritative server under which control of the service provider. Thus the transaction of modification is localized to the related service providers' domains.

Req. #5.: As described in Sec. 3.3, DNS is a feasible way to avoid centralization. M2DMRT does not rely on any new global architecture that could be failure points.

5.4 Evaluation against Feasibility Requirements

This section evaluates M2DMRT against feasibility requirements (Req. #6. – 7.).

Req. #6.: In vertically-chained environments, each service provider needs to have as many declaration records as the couplings of end nodes. As discussed in Sec. 5.2 and Sec. 5.4, although an increase of the number of end node couplings affects the processing time for modifications, it can be considered minimal since the end user is not impacted directly.

Req. #7.: In horizontally-chained environments, declaration records will be made by each set of service providers that serve end users a redirect, which is enough for end users to validate all services in the chain (and therefore the overall service as a whole). In the EC – payment – 3-D Secure example in Figure 1, each assurance between EC – payment and payment – 3-D Secure is enough for the entire service integrity assurance.

Req. #8.: As a result of the experiments in Sec. 5.2, the average time for registration process is evaluated to be sufficiently short, as this process runs only during initial registration or when a record modification becomes necessary.

Req. #9.: The declaration records are published only from the reroute origin service provider's DNS authoritative server by design. This can mitigate the operational cost of the destination service provider, which this paper assumes to have many relationships with different reroute origin side service providers, as is the case with many PaaS payment providers and cloud services.

5.5 Considerations for Real World Implementation and Deployment

From the view point of the entire Internet architecture, M2DMRT requires only two additional DNS resource records per relationship by taking advantage of the existing DNS environment. M2DMRT is believed to be feasible enough to support the current Internet architecture. However, there still are some consideration for implementation in real world.

From the service providers' perspective, they are required to protect their DNS zones with DNSSEC; however, there still are non DNSSEC-compliant DNS zones which are not ready for M2DMRT deployment as service providers.

On the other hand, from the end user's perspective, their full service resolvers or the clients themselves need to validate DNSSEC chain when fetching declaration record. Similar to the DNS zone authoritative servers, there still are non DNSSEC validating full service resolvers. If the end user have no obstacle to query and validate DNSSEC chain by themselves using clients on their network, they can validate M2DMRT declaration record without relying full service resolvers.

When a declaration record is registered, modified, or deleted, the client must be able to detect the change. However, if there are stale caches on a full service resolver or stub resolver used by any of the end users, the validation process may be impacted adversely. Service providers can set shorter TTL for each declaration record RRset to minimize the impact.

6. Conclusion

This paper proposed M2DMRT, a lightweight and self-manageable mutual declaration mechanism of multi-provider relationship, by signing TLS public keys mutually and publishing declaration records in DNSSEC-protected DNS zones. The quantitative evaluation shows that M2DMRT is feasible sufficiently from the perspective of processing speed. The qualitative evaluation shows that M2DMRT satisfies all five of this paper's design requirements, demonstrating the proposed mechanism can prevent attacks (threats) which the current TLS security model does not cover.

This paper concludes that the proposed M2DMRT has no significant negative effect to the Internet environment including service providers rerouting traffic and end users using such providers. Furthermore, M2DMRT may help facilitate development of a more trustworthy Web by introducing the concept of redirect integrity assurance, which has not been discussed previously.

This paper leaves consideration of the effects of stale DNS cache on the end user's client and implementation of a client that enables end users to verify declaration records as future works.

References

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," IETF, Tech. Rep. RFC 8446, 2018.
[2] D. K. Gillmor, "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)," IETF, Tech. Rep. RFC 7919, 2016.
[3] Y. Nir, S. Josefsson, and M. Pégourié-Gonnard, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier," Internet Engineering Task Force, Request for Comments RFC 8422, 2018.
[4] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," IETF, Tech. Rep. RFC 5280, 2008.
[5] WHATWG, "Fetch Standard." https://fetch.spec.whatwg.org/#cors-protocol
[6] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," IETF, Tech. Rep. RFC 6797, 2012.
[7] M. Marlinspike, "New Tricks For Defeating SSL In Practice," in *Proc. of BLACKHAT Europe '09*, 2009.
[8] X. Li, C. Wu, S. Ji, Q. Gu, and R. Beyah, "HSTS Measurement and an Enhanced Stripping Attack Against HTTPS," in *Security and Privacy in Communication Networks*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, X. Lin, A. Ghorbani, K. Ren, S. Zhu, and A. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 489–509.
[9] C. Kerschbaumer, T. Yavor, J. Gaibler, and A. Edelstein, "Firefox 91 introduces HTTPS by Default in Private Browsing." https://blog.mozilla.org/security/2021/08/10/firefox-91-introduces-https-by-default-in-private-browsing
[10] "Increasing HTTPS adoption." https://blog.chromium.org/2021/07/increasing-https-adoption.html
[11] "HSTS Preload List Submission." https://hstspreload.org/
[12] J. Selvi, "Bypassing HTTP Strict Transport Security," in *Proc. of BLACKHAT Europe '14*, 2014.
[13] M. Zhang, X. Zheng, K. Shen, Z. Kong, C. Lu, Y. Wang, H. Duan, S. Hao, B. Liu, and M. Yang, "Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks," in *Proc. of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1939–1952.
[14] "Incorrect proxying of 24 hostnames on January 24, 2022," 2022. http://blog.cloudflare.com/incorrect-proxying-of-24-hostnames-on-january-24-2022/
[15] "3-D Secure - EMVCo." https://www.emvco.com/emv-technologies/3d-secure/
[16] S. M. Z. U. Rashid, M. I. Kamrul, and A. Islam, "Understanding the Security Threats of Esoteric Subdomain Takeover and Prevention Scheme," in *Proc. of 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, 2019, pp. 1–4.
[17] M. Squarcina, M. Tempesta, L. Veronese, S. Calzavara, and M. Maffei, "Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 2917–2934.
[18] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," IETF, Tech. Rep. RFC 6962, 2013.
[19] "Same Origin Policy - Web Security." https://www.w3.org/Security/wiki/Same_Origin_Policy
[20] D. Akhawe, F. Braun, F. Marier, and J. Weinberger, "Subresource Integrity," W3C, REC-SRI-20160623, 2016.
[21] Y. Liu, Y. Zhang, S. Zhu, and C. Chi, "A Comparative Study of Blockchain-Based DNS Design," in *Proc. of the 2019 2nd International Conference on Blockchain Technology and Applications*. Xi'an China: ACM, 2019, pp. 86–92.
[22] K. Matsuoka and T. Suzuki, "Blockchain and DHT Based Lookup System Aiming for Alternative DNS," in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, 2020, pp. 98–105.
[23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.
[24] H. Liu, X. De Foy, and D. Zhang, "A multi-level DHT routing framework with aggregation," in *Proc. of the second edition of the ICN workshop on Information-centric networking - ICN '12*. ACM Press, 2012, p. 43.
[25] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Decentralized Business Review*, 2008.
[26] "Handshake." https://handshake.org
[27] S. Rose, M. Larson, D. Massey, R. Austein, and R. Arends, "DNS Security Introduction and Requirements," IETF, Tech. Rep. RFC 4033, 2005.
[28] R. M. Gieben and M. Mekking, "Authenticated Denial of Existence in the DNS," IETF, Tech. Rep. RFC 7129, 2014.
[29] P. E. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," IETF, Tech. Rep. RFC 6698, 2012.
[30] P. A. Vixie, S. Thomson, Y. Rekhter, and J. Bound, "Dynamic Updates in the Domain Name System (DNS UPDATE)," IETF, Tech. Rep. RFC 2136, 1997.
[31] F. Dupont, S. Morris, P. A. Vixie, D. E. Eastlake 3rd, O. Guomundsson, and B. Wellington, "Secret Key Transaction Authentication for DNS (TSIG)," IETF, Tech. Rep. RFC 8945, 2020.