Formal Modeling and Verification for Domain Validation and ACME

Karthikeyan Bhargavan[1], Antoine Delignat-Lavaud[2] and Nadim Kobeissi[1]

[1] INRIA
{karthikeyan.bhargavan, nadim.kobeissi}@inria.fr
[2] Microsoft Research
antdl@microsoft.com

Abstract. Web traffic encryption has shifted from applying only to sensitive websites (such as banks) to a majority of all Web requests. Until recently, one of the main limiting factors for enabling HTTPS was the requirement to obtain a valid certificate from a trusted certification authority. This process traditionally involves steps such as paying a certificate issuance fee, ad-hoc private key and certificate request generation, and domain validation procedures. To remove this barrier of entry, the Internet Security Research Group (ISRG) introduced "Let's Encrypt", a new non-profit certificate authority that uses a new protocol called Automatic Certificate Management Environment (ACME) to automate certificate management at all levels (request, validation, issuance, renewal, and revocation) between clients (website operators) and servers (certificate authority nodes). Let's Encrypt's success is measured by its issuance of over 27 million free certificates since its launch in April 2016. In this paper, we survey the existing process for issuing domain-validated certificates in major certification authorities. Based on our findings, we build a security model of domain-validated certificate issuance. We then model the ACME protocol in the applied pi-calculus and verify its stated security goals against our security model. We compare the effective security of different domain validation methods and show that ACME can be secure under a stronger threat model than that of traditional CAs. We also uncover weaknesses in some flows of ACME 1.0 and propose verified improvements that have been adopted in the latest protocol draft submitted to the IETF.

1 Introduction

Since the dawn of HTTPS, being able to secure a public website with SSL or TLS requires obtaining a signature for the website's public certificate from a certificate authority [1] (CA). All major operating system and browser vendors maintain lists of trusted CAs (represented by their root certificates) that can legitimately attest for a reasonable link between a certificate and the identity of the server or domain it claims to represent.

For example, all major operating systems and browsers include and trust Symantec's root certificates, which allows Alice to ask Symantec to attest that the certificate she uses on her website AliceShop.com has indeed been issued to her, rather than to an attacker trying to impersonate her website. After Alice pays Symantec some verification fee, Symantec performs some check to verify that Alice and her web server indeed have the authority over AliceShop.com. If successful, Symantec then signs a certificate intended for that domain. Since the aforementioned operating systems already trust Symantec, this trust now extends towards Alice's certificate being representative of AliceShop.com.

The security of this trust model has always relied on the responsibility and trustworthiness of the CAs themselves, since a single malicious CA can issue arbitrary valid certificates for any website on the Internet. Each certificate authority is free to engineer different user sign-up, domain validation, certificate issuance and certificate renewal protocols of its own design. Since these ad-hoc protocols often operate over weak channels such as HTTP and DNS without strong cryptographic authentication of entities, most of them can be considered secure only under relatively weak threat models, reducing user credentials to a web login, and domain validation to an email exchange.

The main guidelines controlling what type of domain validation CAs are allowed to apply are the recommendations in the CA/Browser Forum Baseline Requirements [2]. These requirements, which are adopted by ballot vote between the participating organizations, cover the definition of common notions such as domain common names (CNs), registration authorities (RAs) and differences between regular domain validation (DV) and extended validation (EV).

These guidelines have not proven sufficient for a well-regulated and well specified approach for domain validation: Mozilla was recently forced to remove WoSign [3] (and its subsidiary StartSSL, both major certificate authorities) from the certificate store of Firefox and all other Mozilla products due to a series of documented instances that range from the CA intentionally ignoring security best-practices for certificate issuance, to vulnerabilities allowing attackers to obtain signed certificates for arbitrary unauthorized websites.

The lack of a standardized protocol operating under a well-defined threat model and with clear security goals for certificate issuance has so far prevented a systematic treatment of certificate issuance using well-established formal methods. Instead, academic efforts to improve PKI security focus on measurement studies [4, 5] and transparency and public auditability mechanisms [6, 7] for already-issued certificates.

In 2015, a consortium of high-profile organizations including Mozilla and the Electronic Frontier Foundation launched "Let's Encrypt" [8], a non-profit effort to specify, standardize and automate certificate issuance between web servers and certificate authorities, and to provide certificate issuance itself as a free-of-charge service. Since its launch in April 2016, Let's Encrypt has issued more than 27 million certificates [9] and has been linked to a general increase in HTTPS adoption across the Internet.

Let's Encrypt also introduces ACME [10], an automated domain validation and certificate issuance protocol that gives us for the first time a protocol that can act as a credible target for formal verification in the context of domain validation. ACME also removes almost entirely the human element from the process of domain validation: the subsequently automated validation and issuance of millions of certificates further increases the necessity of a formal approach to the protocol involved.

In this paper, we formally specify, model and verify ACME using ProVerif, an automated protocol verifier [11] that operates in the symbolic model and accepts protocol representations in the applied-pi calculus. Against a classic symbolic protocol adversary, ACME achieves most of its stated security goals. Notably, we show that ACME's design allows it to resist a substantially stronger threat model than the ad-hoc protocols of traditional CAs that rely on bearer tokens (passwords, cookies, authorization strings) for authentication and domain validation, thanks to its stronger cryptographic credentials and to the binding between the clients identity and the validated domain.

Nevertheless, we still discover issues and weaknesses in ACME's domain validation and account recovery features, potentially amounting to user account compromise. We attempt to address in this paper what seem to be open questions regarding ACME: how does ACME compare to the existing security model of the actual top real-world certificate authorities? How can we most fruitfully illustrate and formally verify its security properties, and what can we prove about them?

Contributions Our contributions in this paper consist of:

- **A survey of the domain validation practices of current CAs:** in §2, we survey the issuance process and infrastructure of 10 of the most popular certificate authorities. We observe that traditional CAs support multiple methods for assessing domain control, that rely on different security assumptions.
- **A threat model for domain validation:** in §3, we specify a high-level threat model for certificate issuance based on domain validation, which applies both to both traditional CAs and ACME. We relate this threat model to the various domain control validation methods surveyed in §2. In §4, we demonstrate that ACME resists a stronger threat model than other CAs.
- **Formally specifying and verifying ACME:** in §4, we formally specify the ACME protocol within a symbolic model in the applied π-calculus that encodes the adversarial capabilities described in §3. We verify the main security goals of ACME using the ProVerif model checker. Although ACME is shown to be more resistant to attacks than ad-hoc CAs, we also discover weaknesses in ACME's domain validation and account recovery and propose countermeasures.

2 Current State of Domain Validation

A goal of this paper is to establish a relationship between current domain validation practices in the real world and a more formal threat model on which we base our security results. We begin by taking a closer look into the network infrastructure, user authentication and domain validation protocols currently in use by traditional CAs.

Our panel of surveyed CA is selected from the data set of Delignat-Lavaud et al. [5], which uses machine learning to classify certificates issued by domain validation. Our CA panel covers about 85% of the collected domain validated certificates from 2014, which is consistent with the January 2015 market share data from the Netcraft SSL survey[3]. For each CA, we obtain a regular one year, single-domain certificate signature for a domain name that we own.

§3.2.2.4 of the CA/Browser Forum's Baseline Requirements allow for domain validation to occur in ten different ways, including over postal mail. Of these methods, only three are in popular use: validation via email, the setting of an arbitrary DNS record, or serving some HTTP value on the target domain.

2.1 Domain Validation Mechanisms

With ad-hoc CAs, user C authenticates its identity C_{pk} to CA A as a simple username/password web login, with an option for account recovery via email. C can then request that A validate some domain $C_w \subset C_w^\star$, where $C_w^\star$ is the set of all domain names that C controls. A's flow with the various domain validation channels proceeds thus:

- *HTTP Identifier* A sends to C a nonce A_{N^C} via an HTTPS channel that C must then advertise at some agreed-upon location under C_w. A then accesses C_w using an unauthenticated, unencrypted HTTP connection to ensure that it can retrieve A_{N^C}. This identifier depends *both* on honest DNS resolution of the validated domain's A/A6 records and an untampered HTTP connection to the domain.
  In practice, we find that CAs that allow HTTP identifiers require the nonce to be written on a text file with a long random name in the root of the validated domain. An attacker able to respond to HTTP requests for such names may get a certificate without access to the domain's DNS records.
- *DNS identifier* A sends to C a nonce A_{DNS^C} via an HTTPS channel that C must then advertise at some agreed-upon `TXT` record under the DNS records of C_w. A then queries C_w's name servers using to ensure that it can retrieve A_{DNS^C}. This identifier is dependent on honest resolution of the TXT record. None of the CAs we surveyed advertises DNSSEC support to ensure this DNS resolution is indeed authentic. As an experiment, we set up a DNSSEC-enabled domain and configured our nameserver to send an invalid RRSig for the TXT record of the domain validation nonce for Comodo. The validation ultimately completed, indicating that the use of DNSSEC does not currently prevent attacks against DNS-based domain validation by current CAs.
- *Email identifier* A sends to C a URI A_{URI^C} via an email to an address E_C that A presumes to belong to C. Accessing this URI causes A to issue the certificate for C_w. This identifier is dependent on the confidential transport

[3] `https://www.netcraft.com/internet-data-mining/ssl-survey/`

of the email (which may be routed through third party SMTP servers that are not guaranteed to use TLS encryption) and honest DNS resolution of the validated domain's MX records.

In practice, we observe that CAs use dangerous heuristics to generate a list of possible E_C that C can pick from: first, they presume that any email addresses that appear in the WHOIS records of C_w is controlled by C. A large majority of registrars provide WHOIS privacy services to defend against spam. Such services can easily obtain certificates for any of their customers' domains as validation email transit through their mail servers. Second, CAs' heuristics include generic names such as `postmaster, webmaster`, or `admin`. If the validated domain provides an email service for which users may chose their username, an attacker may register under one of those generic names and obtain a unauthorized certificate. Such attacks have been carried successfully in the past against public email services such as Hotmail.

Once one of the above identifiers succeeds in validating C's ownership of C_w to A, A issues the certificate and the protocol ends.

2.2 User Authentication and Domain Validation

While CAs are required to document their certificate issuance policies in Certificate Practice Statements [12–19], we find that these statements are not always accurate or complete (for instance, they typically provision for validation methods that are not offered in practice; the address heuristics for email-based validation is rarely listed exhaustively). Most ad-hoc CAs in our study favor email-based validation. Unlike HTTP and DNS identifiers, email identifiers effectively rely on a *read* capability challenge instead write access proof for C. In §3, we discuss how email identifiers are the weakest available form of identification given our threat model. In §4, we elaborate on a weakness in ACME affecting both account recovery and domain validation. While this weakness is also generalizable to traditional certificate authorities, ACME offers an opportunity for a stronger fix.

None of CAs we surveyed offers a login mechanism that is completely independent of email. An exception almost occurs with StartSSL, which supports browser-generated X.509 client certificates for web login, but this exception is negated by the email-based account recovery in case of a lost certificate private key. Reliance on the security of the email channel can in many cases be even more serious: in many surveyed CAs, simply being able to complete a web login enables user to re-issue certificates for domains they had already validated before, without further validation.

A scan of the DNS MX and NS records of the web's top 10,000 websites (according to Alexa.com) [20] showed that roughly 45% of surveyed domain names used only six DNS providers, of which CloudFlare alone had a 18% share.[4] A similar centralization of authority exists with email, where the top six providers

[4] CloudFlare incidentally also operates Let's Encrypt's infrastructure, rendering it a centralized point of failure for ACME and ad-hoc CAs alike. While ACME is

CA	**Identifiers**	**Email Recovery**	**Public Key Auth.**	**Per-CSR Check**
AlphaSSL	Email	✓	✗	N/A
Comodo PositiveSSL	Email	✓	✗	✓
DigiCert	Email	✓	✗	✗
GeoTrust QuickSSL	Email	✓	✗	✗
GlobalSign	HTTP, DNS, Email	✓	✗	✗
GoDaddy SSL	HTTP, DNS	✓	✗	✗
Let's Encrypt (ACME draft-1)	HTTP	✓	✓	✗
Network Solutions	Email	✓	✗	✗
RapidSSL	Email	✓	✗	✓
SSL.com BasicSSL	HTTP, DNS, Email	✓	✗	N/A
StartCom StartSSL	Email	✓	✓	✗

Fig. 1: Popular CAs, their validation methods, whether they permit account recovery via email, whether they allow login via a public-key based approach (such as client certificates) and whether domain validation is carried out once for every certificate request, even for already-validated domain names.

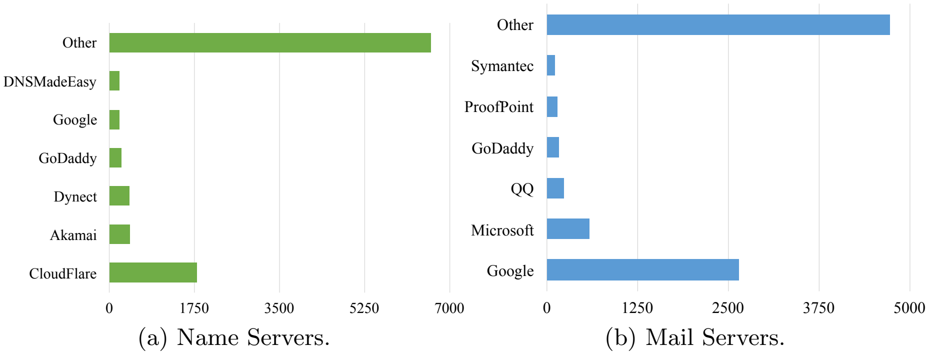
serve more than 55% of domain names surveyed, with Google alone holding roughly 27% market share (Figure 2.)

These results suggest that the number of actors of which the compromise could affect traditional domain validation is significantly small. This is relevant given how top CAs allow for account recovery, certificate re-issuance and more with simple email-based validation.

3 A Security Model for Domain Validation

The protocols considered in this paper operate between a party C claiming to serve and represent one or more domain names C_w (for which it wants certificates), and it is incumbent upon a certificate issuer A to verify that all domains in C_w are indeed controlled and managed by C. User C authenticates itself to the certificate authority A using a public key C_{pk} of a private identity C_k. C can

a centralization-agnostic protocol, Let's Encrypt operates with a fully centralized infrastructure.



(a) Name Servers.

(b) Mail Servers.

Fig. 2: Provider repartition among the Alexa Top 10,000 global sites, as of October 2016. Notably, CloudFlare and Akamai also provide CDN services to domains under their name servers, allowing them stronger control over HTTP traffic.

then link identifiers under C_k that prove that it manages and controls domains in C_w.

This and following sections are largely based on our full symbolic model[5] of ACME and ad-hoc CA protocol and network flow, which is written in the applied pi calculus and verified using ProVerif. Excerpts of this model are inlined throughout.

3.1 Security Goals and Threat Model

Our security goals are straightforward: for any domain $C_w \in C_w$, A must not sign a certificate asserting C's ownership of C_w for that domain unless C can validate C_{pk} as representing the identity that owns and manages this domain. ACME allows C to validate C_w with respect to C_{pk} by using the secret value C_k in order to demonstrate either read or write capabilities on certain predefined network channels, each with its own security model. A domain name C_w is considered *validated* under C_w if C_k can be used to complete a verification challenge on one of the network channels offered by the ACME protocol between C and A that in consequence asserts a relationship between C_{pk} and C_w.

The network topology, channels and actors are essentially the same for both ACME and ad-hoc CAs. However, the manner in which these actors communicate over the channels is different, and leads to different attempts to establish the same security guarantees.

Channels Intuitively, the channels we want encapsulate the following properties:

[5] Full model available at https://github.com/Inria-Prosecco/acme-model

- **HTTPS Channel** Intuitively a regular web channel, we treat it as a A-authenticated duplex channel whereupon anyone can send a request to A, only A can read this request and respond, and only the sender can read this response.
- **Strong Identifier Channels** These channels must be assumed to be writable only by C. They are therefore relevant for HTTP and DNS Identifiers.
- **Weak Identifier Channel** Anyone can write to this channel, but only C can read from it. This makes it relevant for domain validation via email identifiers.

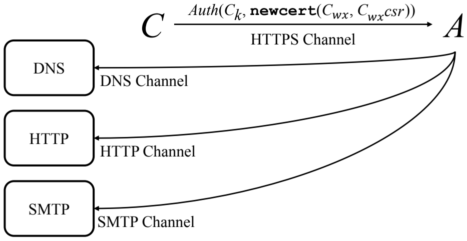


Fig. 3: Channels overview.

A shared consideration between ACME and ad-hoc CAs involves the critical importance of DNS resolution: if the attacker can simply produce false DNS responses for A resolving a domain request for any domain in C_w, it becomes impossible to safely carry out domain validation under any circumstances. As a sidenote, this allows us to argue that since the DNS channel must be trusted, it could also be considered as the safest channel on which to carry out domain validation using DNS Identifiers since that would allow C to avoid needlessly involving other channels.

In formally describing our network model in ProVerif, we simulate simultaneous requests from Alice, Bob and Mallory as independent clients C. Alice and Bob both act as honest clients, while Mallory acts as a compromised participant client. All three follow the same protocol top-level process. We also simulate two independent ACME CAs, which interchangeably assume the role of A. For each C, we specify a triple of distinct channels:

$$(C_{HTTP}, C_{EMAIL}, C_{DNSTXT})$$

Each channel represents access to a different domain validation mechanism. While C is given complete access over these channels, the channels are only handed to C_{EMAIL} is only handed to A after being applied through a "write transformation" which returns a variant of the channel that is effectively write-only:

$$w(channel) \rightarrow channel$$

Similarly, a "read transformation" $r(channel) \rightarrow channel$ is applied to C_{HTTP} and C_{DNSTXT}.

A routing proxy is then specified in order to model the transportation across these channels by executing the following unbounded processes in parallel[6]:

[6] We also specify a fully public channel named `pub`.

$$in(w(C_{EMAIL}), x);\ out(r(C_{EMAIL}), x)$$
$$in(pub, x);\ out(r(C_{EMAIL}), x)$$
$$in(w(C_{HTTP}), x); out(pub, x); out(r(C_{HTTP}), x)$$
$$in(w(C_{DNSTXT}), x); out(pub, x); out(r(C_{DNSTXT}), x)$$

Threat Model We assume that the adversary controls parts of the network and so can intercept, tamper with and inject network messages. As such, an attacker could make requests for domains they do not own, intercept and delay legitimate certificate requests, and so on. Our adversary has full access to *pub*, $w(C_{EMAIL})$, $r(C_{HTTP})$ and $r(C_{DNSTXT})$. We also publish Mallory's channels and C_k over *pub*. As such, the attacker controls a set of valid participants (e.g. M) with their own valid identities (e.g. M_k, M_{pk}). The attacker may advertise any identity for its controlled principals, including false identities, and may attempt to obtain a certificate for domains not legitimately under M_w.

The adversary also has at his disposal certain special functions:

- *PoisonDnsARecord*, which takes in a domain C_w and allows the attacker to poison its DNS records to redirect to a server owned by M. Calling this function triggers the *ActiveDnsAttack*(C_w) event.
- *ManInTheMiddleHttp*, which allows the attacker to write arbitrary HTTP requests as if they were emitting from C_{HTTP} by disclosing C_{HTTP} to the attacker. Calling this function triggers the *ActiveHttpAttack*(C_w) event.

3.2 ProVerif Events and Queries

Under ProVerif, queries under our symbolic model are constructed from sequences of the following events, each callable by a particular type of actor:

- **Client** The client is allowed to assert that they own some domain by triggering the event $Owner(C, C_w)$. Once C receives a certificate $C_{w_{cert}}$ for C_w from A, they also trigger $CertReceived(C_w, C_{w_{cert}}, C_{pk}, A_{pk})$
- **Server** The server (ACME instance or CA) triggers the event $HttpAuth(C_{pk}, C_w)$, $DnsAuth(C_{pk}, C_w)$ and $EmailAuth(C_{pk}, C_w)$ depending on the type of domain validation used. Once A issues a certificate $C_{w_{cert}}$ for C_w to C, they also trigger $CertIssued(C_w, C_{w_{cert}}, C_{pk}, A_{pk})$
- **Adversary** As noted above, the adversary may trigger the events $ActiveDnsAttack(C_w)$ and $ActiveHttpAttack(C_w)$. In addition, the adversary is allowed to masquerade as M in order assert that they own some domain by triggering the event $Owner(M_{pk}, C_w)$.

Queries Queries encode the security properties that we expect our model to satisfy. For example, informally, we expect the a *CertIssued* event may only occur following an *HttpAuth* or *DnsAuth* event for the same domain, expressing the fact that ACME should not issue a certificate for an non-validated domain under any circumstance. Running ProVerif on the query can result in three outcomes: either it diverges (in which case the model or query needs to be simplified), or it proves that the model satisfies the query, or it finds a counter-example and outputs its trace (which can be turned into an attack).

Validation with DNS Identifiers We assert that if DNS validation succeeded, then A must have been able to successfully carry out DNS validation according to spec, *or* an adversary was able to instantiate an active DNS poisoning attack (with no third possible scenario). In ProVerif, this can be expressed using injective event queries:

$$\mathit{DnsAuth}(C_{pk}, C_w) \implies (\mathit{Owner}(C_k, C_w) \vee \mathit{DnsAttack}(C_w))$$

Validation with HTTP Identifiers We explicitly show that HTTP authentication is weaker than DNS authentication, since it is possible under both cases of DNS poisoning *and* an HTTP man-in-the-middle attack:

$$\mathit{HttpAuth}(C_{pk}, C_w) \implies \mathit{Owner}(C_k, C_w) \;\vee\; (\mathit{HttpAttack}(C_w) \vee \mathit{DnsAttack}(C_w))$$

Predictable Certificate Issuance We attempt to verify that all received certificates were issued by the expected CA. This query fails to verify, and leads us to the attack we discuss in §5.2:

$$\mathit{CertReceived}(C_w, C_{w_{cert}}, C_{pk}, A_{pk}) \implies \mathit{CertIssued}(C_w, C_{w_{cert}}, C_{pk}, A_{pk})$$

4 Specifying and Formally Verifying ACME

In this section we provide a formal description of the ACME protocol functionality and identify three issues that affect ACME's security. We also discuss details of how we describe the ACME protocol flow in the applied pi calculus, so that we can verify for certain queries using ProVerif.

4.1 ACME Network Flow

Unlike ad-hoc CAs which are limited to a web login, ACME's authentication depends on C generating a private value C_k and a public signing key C_{pk}, which are used to generate automated client signatures throughout the protocol.

HTTP Identifier A sends to C a nonce A_{NC} via the HTTPS channel. C must then advertise, at an agreed-upon location under C_w, the value (C_{pk}, A_{NC}). A then accesses C_w using an unauthenticated, unencrypted HTTP connection to ensure that it can retrieve the intended value.

ACME also supports a very similar validation mode that operates at the level of the TLS handshake rather than at the HTTP level (using the SNI extension and a specially crafted certificate in place of the HTTP request and response).

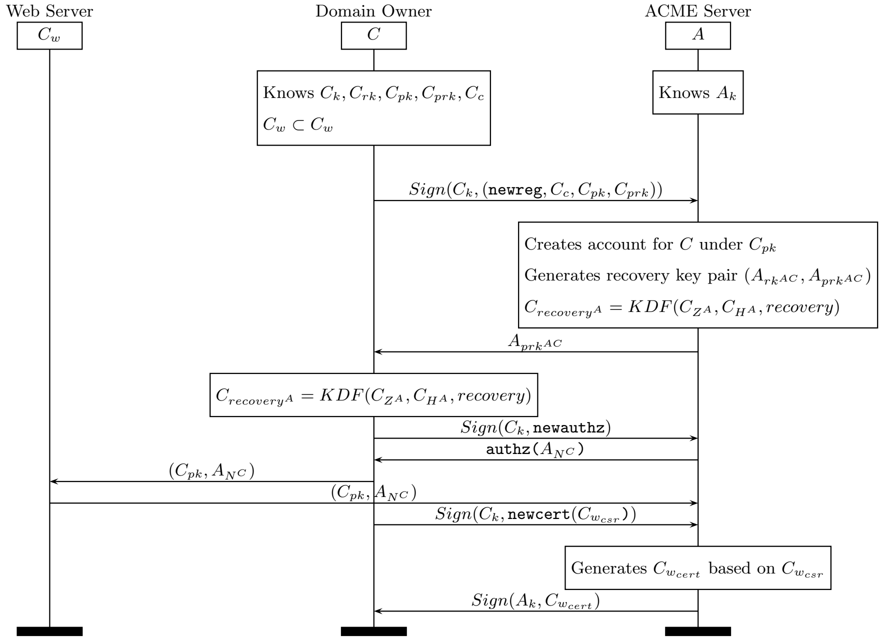

Web Server
C_w
Domain Owner
C
ACME Server
A
Knows $C_k, C_{rk}, C_{pk}, C_{prk}, C_c$
$C_w \subset C_w$
Knows A_k
$Sign(C_k, (\texttt{newreg}, C_c, C_{pk}, C_{prk}))$
Creates account for C under C_{pk}
Generates recovery key pair $(A_{rk^{AC}}, A_{prk^{AC}})$
$C_{recovery^A} = KDF(C_{Z^A}, C_{H^A}, recovery)$
$A_{prk^{AC}}$
$C_{recovery^A} = KDF(C_{Z^A}, C_{H^A}, recovery)$
$Sign(C_k, \texttt{newauthz})$
$\texttt{authz}(A_{N^C})$
(C_{pk}, A_{N^C})
(C_{pk}, A_{N^C})
$Sign(C_k, \texttt{newcert}(C_{w_{csr}}))$
Generates $C_{w_{cert}}$ based on $C_{w_{csr}}$
$Sign(A_k, C_{w_{cert}})$

Fig. 4: ACME draft-1 protocol functionality for C account registration, recovery key generation, and validation with certificate issuance for C_w. This chart demonstrates validation via an HTTP identifier. In draft-3 and above, the HTTP challenge (C_{pk}, A_{N^C}) is replaced with $Sign(C_k, (C_{pk}, A_{N^C}))$.

We believe this mode is intended for TLS termination software and hardware, and despite its apparent complexity, it is semantically equivalent to the HTTP identifier method. Since the details of the formatting of payloads is abstracted in our symbolic model we model both TLS-SNI and HTTP validation under the same framework in our model.

DNS Identifier A sends to C a nonce A_{nonce^C} via the HTTPS channel. C advertises this nonce in the form of a DNS record served by C_w's name servers, thereby proving ownership of C_w. A can then query its DNS server to verify that the nonce has been set. While this behavior is specified in ACME, it is not used in any implementation of Let's Encrypt: since ACME is designed to take advantage of domain validation methods that can be automated and since DNS record management depends on a series of ad-hoc protocols of its own between C and DNS service providers, it is not used by ACME.

Out-of-Band Validation The ACME standard draft supports an out-of-band validation mechanism, which can be used to implement legacy validation methods, including email-based validation. However, since this method is underspecified, we do not cover it in our models and advice against using any out-of-bound validation unless it is analyzed under a specific model.

4.2 ACME Protocol Functionality

In this paper we focus on draft-1 of the IETF specification for the ACME protocol, which is as of October 2016 also the draft specification deployed in official Let's Encrypt client and server implementations. In part due to the issues we discuss in the paper and have communicated with the ACME team, draft-3 (and subsequently draft-4) does away with some features, most notably Account Recovery, and generally is resistant to the issues discussed here.

Preliminaries In some parts of ACME's protocol flow, C and A will need to establish a number of shared secrets, each bound to a strict protocol context, over their public keys. In ACME, this is accomplished using ANSI-X9.63-KDF:

1. C and A agree on a ECDH shared secret C_{Z^A} using their respective key pairs (C_k, C_{pk}) and (A_k, A_{pk}).
2. A hashing function C_{H^A} is chosen according to the elliptic curve used to calculate C_{Z^A}: $SHA256$ for $P256$, $SHA384$ for $P384$ and $SHA512$ for $P521$.
3. $C_{label^A} = KDF(C_{Z^A}, C_{H^A}, label)$, with $label$ indicating the chosen context for this particular key's usage.

As a protocol, ACME provides the following seven certificate management functionalities (illustrated in Figure 4) between web server C and certificate management authority A:

- *Account Key Registration* In this step, C specifies her contact information (email address, phone number, etc.) as C_c and generates a random private signing key C_k with (over a safe elliptic curve) a public key C_{pk}. A `POST` request is sent to A containing $Sign(C_k, (\texttt{newreg}, C_c, C_{pk}))$. The `newreg` header indicates to A that this is an account registration request. If A has no prior record of C_{pk} being used for an account, and if the message's signature is valid under C_{pk}, A creates a new account for C using C_{pk} as the identifier and responds with a success message.
- *MAC-Based Account Recovery* C may choose to identify an account recovery secret with A. In order to do this, C generates an account recovery key pair (C_{rk}, C_{prk}) and simply includes C_{prk} in an optional `recoverykey` field in its initial `newreg` message to A. A generates the complementary $(A_{rk^{AC}}, A_{prk^{AC}})$ and both parties calculate $C_{recovery^A}$ using their recovery key pairs. A communicates $A_{prk^{AC}}$ in its response to C. Later, if C loses C_k, she can ask A to re-assign her account to a new identity $(C_{k'}, C_{pk'})$ by using $C_{recovery^A}$ as a key to generate a MAC of some value chosen by A.

- *Contact-Based Account Recovery* C can request that A send a verification token to one of the contact methods previously specified in C_c. For example, this could be a URI sent to an email in C_c. If C successfully opens this URI, she becomes free to replace C_{pk} with a $C_{pk'}$ for some arbitrary $C_{k'}$ at A.
- *Identifier Authorization* C can validate its ownership of a domain C_w in C_w by providing one of the identifiers discussed in §3 to A. C must first request authorization for C_w by sending a `newauthz` message. A then responds with the types of identifiers it is willing to accept in a `authz` message. C is then free to use any one of the permitted identifiers to validate its ownership of C_w and allow A to sign certificates for it issued to C and tied to the identity C_{pk}.
- *Certificate Issuance and Renewal* After C ties an identifier to C_w under C_{pk}, it may request that a certificate be issued for C_w simply by requesting one from A. Generally, A will send the signed certificate with no further steps required. The renewal procedure is similarly straightforward.
- *Certificate Revocation* C may ask A to revoke the certificate for C_w by sending a `POST` message containing the certificate in question, signed under either C_{pk} or the key pair for the certificate itself. C may choose which key to use for this signature. A verifies that the public key of the key pair signing the request matches the public key in the certificate, and that the key pair signing the request is an account key, and the corresponding account is authorized to act for all of the identifier(s) in the certificate.

Given this description of the ACME protocol and the threat model defined in §3, we modeled ACME using the automated verification tool ProVerif [21]. In our model, we involve three different candidates for C: Alice, Bob and Mallory, and two CA candidates as A.

In our automated verification process, we consider an active attacker over the three channels specified in §3. As a result, we were able to find the issues discussed in §5.2. The first two are relatively minor; however, the third could lead to account compromise in the case of contact-based account recovery, and potentially to the issuance of false certificate signatures if email-based domain validation were to be implemented in ACME. Furthermore, this third issue is also generalizable to affect traditional certificate authorities, as described in §2.

4.3 Model Processes

Using the modeling conventions we established in §3 which include channels, adversaries, actors and events, we instantiate in our ProVerif model of ACME a top-level process that executes the following processes in parallel:

- *ClientAuth* Run simultaneously by Alice, Bob and Mallory assuming the role of C (with a compromised Mallory), this process registers a new account with A and sends the queries illustrated in Figure 4. The events *Owner* and *CertReceived* are triggered as part of this process.

- *ServerAuth* Run simultaneously by two independent CAs assuming the role of A, this process accepts registrations from C and follows the protocol illustrated in Figure 4. The events *HttpAuthenticated* and *CertIssued* are triggered as part of this process.

The processes *RoutingProxy*, *PoisonDnsARecord* and *ManInTheMiddleHttp*, all described in §3, are also run in parallel with the above.

5 Analysis Results

5.1 Weaknesses in Traditional CAs

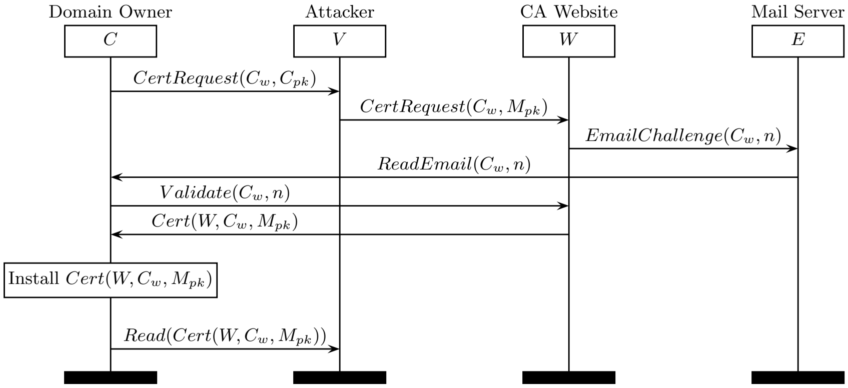


Fig. 5: Attack on Email Validation: Concurrent Request by Active Adversary.

Traditional CA dependence on weak channels gives us a threat model where real-world attacks can have a small cost and come with severe consequences.

Email Validation In ad-hoc CAs, C is generally simply sent an email containing a URI to their email inbox, which they're supposed to click in order to validate for their chosen domain. Figure 5 shows an attack rendered possible by this mechanism. A could instead, upon a validation request, redirect C's browser to a secret, nonce-based URI A_{URI^C} served to C over the HTTPS channel, and independently mail C the value $HMAC(A_{hk}, A_{URI^C})$ for some secret A_{hk} held by A. C would need to retrieve this second value and enter it inside the page at A_{URI^C}. This approach would largely negate the weakness discussed in §5.2, since an attacker-induced validation email would result in an email that does not include a value matching the URI given by A to C at the beginning of the validation process.

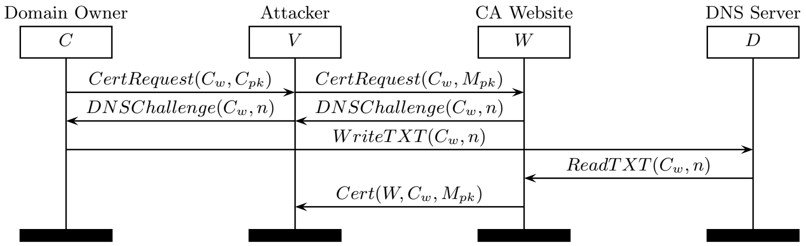


Fig. 6: Active attack on DNS/HTTP/Email Validation when using just nonces.

Usage of Nonces Traditional CAs use random nonces with no special cryptographic properties as the values that they then verify over HTTP, email or DNS. In addition to this helping caused the attack described above, another more general attack on nonces is shown in Figure 6 in the case of an active attacker. For example, this attack can be used by a compromised CA website to get certificates issued for domain C_w by another (more reputable) CA, hence amplifying the compromise across CAs. None of these attacks would be effective if nonces were tied to some cryptographic properties, such as MACs or even just by deriving them from a hash of the certificate request's public key.

In order to avoid a similar attack, ACME draft-3 and draft-4 require that HTTP identifiers be validated by broadcasting $Sign(C_k, A_{N^C})$ via the web server instead of ACME draft-1's (C_{pk}, A_{N^C}).

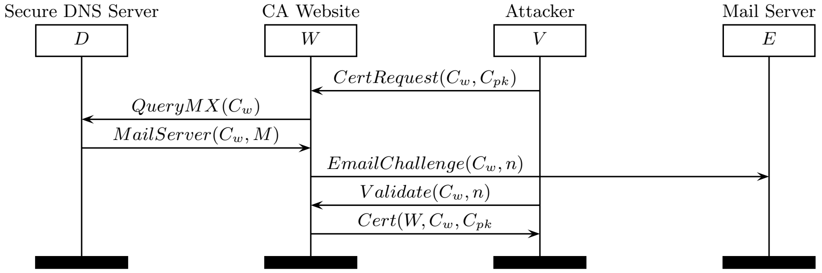


Fig. 7: Attack on Email Validation: Passive Adversary on Email Channel

5.2 Weaknesses in ACME

Cross-CA Attacks on Certificate Issuance Suppose an ACME client C requests a certificate from A, and but A is malicious or the secure channel between A and C is compromised. Now, an attacker can intercept authorization and certificate requests from C to A, and instead forward them to another ACME server A'. If A' requests domain validation with a token T, the attacker forwards the token to the client, who will dutifully place its account key K and token T on its validation channel. A' will check this token and accept the authorization and issue a certificate that the attacker can forward to C.

This means that C asked for a certificate from A, but instead received a certificate from A'. Moreover, it may have paid A for the service, but A' might have done it for free. This issue, while not critical, can be prevented if C checks the certificate it gets to make sure it was issued by the expected CA. An alternative, and possibly stronger, mitigation would be for ACME to extend the Key Authorization string to include the CAs identifier.

More generally, this issue reveals that ACME does not provide channel binding, and this appears as soon as we model the ACME HTTPS Channel. We would have expected to model this as a mutually-authenticated channel since the client always signs its messages with the account key. However, although the clients signature is tunnelled inside HTTPS, the signature itself is not bound to the HTTPS channel. This means that a message from an ACME client C to A can be forwarded by A to a different A' (as long as C supports both A and A'). This kind of credential forwarding attack can be easily mitigated by channel binding. For example, ACME could rely on the Token Binding specifications to securely bind the client signature to the underlying channel. Alternatively, ACME could extend the signed request format to always include the servers name or certificate-hash, to ensure that the message cannot be forwarded to other servers.

Contact-Based Recovery Hijacking While the use of sender-authenticated channels in ACME seems to be relatively secure, more attention needs to be paid to the receiver-authenticated channels. For example, if the ACME server uses the website administrator's email address to send the domain validation token, a naïve implementation of this kind of challenge would be vulnerable to attack.

In the current specification, the contact channel (typically email) is used for account recovery when the ACME client has forgotten its account key. We show how the careless use of this channel can be vulnerable to attack, and propose a countermeasure. Suppose an ACME client C issues an account recovery request for an account under C_{pk} with a new key $C_{k'}$ to the ACME server A. A network attacker M blocks this request and instead sends his own account recovery request for the account under C_{pk} (pretending to be C) with his own key $M_{k'}$. A will then send C an email asking to click on a link. C will think this is a request in response to its own account recovery request and will click on it. Similarly to the (slightly different) flow described in Figure 5, A will think that C has confirmed account recovery and will transfer the account under C_{pk} to the attackers key $M_{k'}$. In the above attack, the attacker did not need to compromise the contact channel (or for that matter, the ACME channel).

The key observation here is that on receiver-authenticated channels (e.g. email) the receiver does not get to bind the token provided by A with its own account key. Consequently, we need to add a further check. The email sent from A to C should contain a fresh token in addition to Cs new account key. Instead of clicking on the link (out-of-band), C should cut and paste the token into the ACME client which can first check that the account key provided by A matches the one in the ACME client and only then does it send the token back to A, or alternatively that the email recipient at C visually confirms that the account key (thumbprint) provided by A matches the one displayed in the ACME client.

The attack described here is on account recovery, but a similar attack appears if we allow email-based domain validation. A malicious ACME server or man-in-the-middle can then get certificate issued for C's domains with its own public key, without compromising the contact/validation channel. The mitigation for that attack would be very similar to the one proposed above.

6 Conclusion

In this paper, we have provided the results of an empirical case study that allowed us to describe a real-world threat model governing both traditional certificate authorities and ACME in terms of user authentication and domain validation. We formally modeled these protocols and provided the results of security queries under our threat model, using automated verification. As a result of our disclosures to the ACME team, the latest ACME protocol version (draft-4) has been designed to avoid the pitfalls that make these attacks possible.

Given the weak threat model that traditional CAs are assuming for their domain validation process, we are not surprised by the regular occurrences of unauthorized certificate issuance (e.g. StartCom in 2008, Comodo and DigiNotar in 2011, WoSign in 2016). We advocate the CA/Browser forum to eventually mandate the use of ACME (or some other well-defined domain validation protocol that can be formally analyzed) to all certification authorities, as a long-term solution to reduce unauthorized certificate issuance. Until the issuance process for the whole PKI is unified, techniques to improve the validation of certificates such as certificate transparency [6] remain necessary to detect issuance failures, and technologies such as DNSSEC [22], DANE [23], or SMTPS may help strengthen the channels involved in legacy domain validation.

References

1. S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 Public Key Infrastructure: Certificate Policy and Certification Practices Framework. RFC 3647, Internet Engineering Task Force, November 2003.
2. CA/Browser Forum. Baseline requirements for the issuance and management of policy-trusted certificates, v.1.1.5, May 2013.

3. Gervase Markham, Ryan Sleevi, Richard Barnes, and Kathleen Wilson. Wosign and startcom.
4. Olivier Levillain, Arnaud Ébalard, Benjamin Morin, and Hervé Debar. One year of SSL Internet measurement. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 11–20, New York, NY, USA, 2012. ACM.
5. Antoine Delignat-Lavaud, Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, Yinglian Xie, and Microsoft Research. Web pki: Closing the gap between guidelines and practices. In *NDSS*, 2014.
6. Google. Certificate transparency.
7. David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 382–393, New York, NY, USA, 2014. ACM.
8. Internet Security Research Group. Let's encrypt overview, 2016.
9. Internet Security Research Group. Let's encrypt statistics, 2016.
10. Richard Barnes, Jacob Hoffman-Andrews, and James Kasten. Automatic certificate management environment (acme), Jul 2016.
11. Bruno Blanchet, Ben Smyth, and Vincent Cheval. Proverif 1.90: Automatic cryptographic protocol verifier, user manual and tutorial, 2014.
12. Comodo CA Ltd. Comodo Certification Practice Statement. Technical report, Comodo CA Ltd., August 2015.
13. DigiCert. DigiCert Certification Practices Statement. Technical report, DigiCert, September 2016.
14. GeoTrust Inc. GeoTrust Certification Practice Statement. Technical report, GeoTrust, September 2016.
15. GlobalSign CA. GlobalSign CA Certification Practice Statement. Technical report, GlobalSign CA, August 2016.
16. Internet Security Research Group. Certification Practice Statement. Technical report, Internet Security Research Group, October 2016.
17. Symantec Corporation. Symantec Trust Network (STN) Certification Practice Statement. Technical report, Symantec Corporation, September 2016.
18. StartCom CA Ltd. StartCom Certificate Policy and Practice Statements. Technical report, StartCom CA Ltd., September 2016.
19. LLC Network Solutions. Network Solutions Certification Practice Statement. Technical report, Network Solutions, LLC, September 2016.
20. Alexa Internet Inc. Top 1,000,000 sites (updated daily), 2013.
21. R. K usters and T. Truderung. Using ProVerif to analyze protocols with Diffie-Hellman exponentiation. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 157–171, 2009.
22. Giuseppe Ateniese and Stefan Mangard. A new approach to dns security (dnssec). In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 86–95. ACM, 2001.
23. Paul Hoffman and Jakob Schlyter. The dns-based authentication of named entities (dane) transport layer security (tls) protocol: Tlsa. Technical report, 2012.