The Web's PKI: An Expository Review and Certificate Validation Cost Simulation

Randy Heiland
Center for Applied Cybersecurity Research
Indiana University
heiland@iu.edu

William C. Garrison III
Dept. of Computer Science
University of Pittsburgh
bill@cs.pitt.edu

Yechen Qiao
Dept. of Computer Science
University of Pittsburgh
yeq1@cs.pitt.edu

Adam J. Lee
Dept. of Computer Science
University of Pittsburgh
adamlee@cs.pitt.edu

Von Welch
Center for Applied Cybersecurity Research
Indiana University
vwelch@iu.edu

ABSTRACT

Public key infrastructure (PKI), comprised of X.509 certificates (PKIX) and cryptographic protocols, helps ensure secure communications for the Web. The creation of PKI is a fascinating story that dates back to the 1970s and came about thanks to people with a strong desire to democratize privacy and security on the Internet. PKIX became the model implementation for PKI that included a fundamental, openly defined, digital certificate. This paper offers an overview of PKIX for a general audience. It also encourages students to explore some of these ideas, with a discussion on modeling and simulation of costs associated with certificate validation, and some computational number theory.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

Keywords

cybersecurity, education, PKI, X.509 certificates, modeling, simulation, cryptography

1. INTRODUCTION

Millions of people use the World Wide Web (Web) on a daily basis. They use it for education, social networking, email, banking, shopping, and much more. The majority of users do not give much thought to the underlying security behind the Web. Some may pause to consider the risk of entering their credit card number on a shopping site, for example; however, in general, security is a mystery to them.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

In this paper, we describe some of the basic security concepts and implementations used by the Web. We write for a general audience, but the more technical sections, e.g., modeling and simulation, are intended primarily for students.

Our primary goals are to 1) provide a general introduction to public key infrastructure (PKI) and cryptography, 2) describe some of the implementations for one of the fundamental operations in PKI: certificate validation, and 3) discuss some simple models and simulations of costs associated with these implementations.

Section 2 introduces the reader to PKI, with some historical context. In Section 3, we define some of the main concepts used by the Web's PKI, including X.509 certificates. Section 4 describes some of the current implementations used on the Web for validating certificates. Sections 5 and 6 are primarily intended for students and discuss computational thinking, modeling, and simulation. A brief discussion of related work is in Section 7. Section 8 shifts the focus from certificate validation and offers an overview of public key cryptography and its connection to number theory. Finally, Section 9 offers some concluding remarks.

2. BACKGROUND

The history of PKI and public key cryptography (PKC), as used by the Web today, is a fascinating story. It is an interesting mix of human, technological, and mathematical ingenuity. It is the foundation on which we have connected millions of people around the world and also built a multi-billion dollar e-commerce industry. And while it continues to serve us very well, it is not perfect. As we will discuss in this paper, PKI and cryptography for the Web continue to evolve in interesting ways.

For hundreds of years, it was assumed that if two people wanted to communicate privately, they would need to share a mutually agreed upon secret "key". One of the most often cited examples is the *Caesar cipher*, used by Julius Caesar to send and receive private messages. By taking a standard alphabet and "rotating" (shifting, with wrap-around at the end) the letters by some fixed number, e.g., 13, one could construct an encrypted message by a simple substitution of letters. The secret key for this particular cipher would therefore be "13" (and the implied algorithm would be "rotate the

alphabet").

PKC introduced the remarkable idea of a "public" key, i.e., a key that could be accessible by anyone. It seemed to go against common sense. If someone wanted to share a private message with someone else, why even consider using a publicly known key? The simple answer is that it allowed for communication between *any* two people and avoided the problem of manually exchanging a private key ahead of time. There will still be a private (secret) key in PKC; however, it will remain private to a single individual. PKC and its use of a public-private key pair helped make it possible for two total strangers to securely exchange digital messages over a potentially insecure network. Explanations of how PKC works range from quite simplistic[37] to very formal[17][27].

Levy's *Crypto*[22] captures the colorful history of PKC and PKI that a general audience should understand and enjoy. The book conveys very well the human aspect of these discoveries. It tells how one person's relentless drive to democratize privacy led to the remarkable discovery[1] of practical security on the then nascent Internet (and eventually to the World Wide Web). Whitfield Diffie, after teaming up with Martin Hellman, published their seminal paper[7] in 1976. They proposed a hypothetical "one-way (mathematical) function" as a mechanism to openly reveal an encryption technique without any (practical) way of inverting it to obtain the decryption technique. In less than two years, Rivest, Shamir, and Adleman[31] published their landmark *RSA* (from their initials) paper that proposed a practical implementation for such a one-way function. More will be said about this in Section 8. A short time after the RSA paper appeared, one of Adleman's students[18] published a thesis on the concept of *certificates* - something that would evolve into X.509 certificates, discussed in the next section.

Diffie and Hellman were recognized for their work with the A.M. Turing Award[2], one of the most prestigious awards in computer science, in 2015. Rivest, Shamir, and Adleman received the Turing award in 2002 for their RSA cryptographic system.

3. PKIX OVERVIEW

The PKI we discuss in this paper is the de facto standard used for the Web. It is known as PKIX because it uses *X.509 certificates*, described in more detail below. In the Internet Engineering Task Force (IETF) RFC 5280[5], we learn:

> The goal of the Internet PKI is to meet the needs of deterministic, automated identification, authentication, access control, and authorization functions.

In addition to defining an X.509 certificate, RFC 5280 also describes *certificate revocation lists* (CRLs) and an algorithm for performing *certificate path validation*. We will describe each of these concepts in more detail below. RFC 5280, together with public key cryptography, are essentially the culmination of Diffie and Hellman's 1976 paper.

What does all of this terminology mean for an ordinary user of the Web? When a person uses a Web browser to visit a "secure" site, i.e., a site whose URL is prefixed with **https** (*secure* HTTP protocol), there is a sequence of (mostly hidden) steps taken to help ensure the site is trustworthy and that communication between it and a browser is secure. The secure Web site will be in possession of a unique X.509 certificate (Section 3.1). Using a secure[3] "digital handshake", the browser will receive this certificate from the Web server and try to determine whether it is valid. If it is valid, the browser proceeds to visit the site; if not, it will (should) notify the user that the site is untrustworthy and refuse to visit the site. Some of the reasons for a certificate being invalid are described below.

However, PKIX is not a perfect solution, for multiple reasons. There have been instances, for example, when certificates should have been revoked and were not[41].

We want to make readers aware of some simple visual cues their browsers provide that indicate the level of security (in theory) associated with a Web site. Although different browsers have slightly different cues, typically the URL in the address bar will be prefixed with some sort of small image decorator. Figure 1 lists three different URLs and their respective security decorators. Typically, if a Web site uses HTTPS, it will have a padlock to indicate "secure". There are still many, many Web sites that use HTTP instead of HTTPS. We show the Web site for CNN news as just one example. Google and others[4][33] are trying to make adoption of HTTPS more widespread. If a HTTPS Web site has taken extra effort (and likely paid a premium price), they can be designated as an Extended Validation (EV) site. The decorator will then include both a padlock and the Web's site's organization name preceding the HTTPS, as shown in the Twitter address bar. We will demonstrate later how a user can obtain more detailed certificate information either from the browser or using command line tools. It should be noted that it is also possible to visit a HTTPS Web site, but the padlock decorator be absent, indicating it has, for some reason(s), been deemed insecure. This would be the case, for example, when visiting a site that is streaming content using a browser plugin that has been deemed insecure. Again, a user can easily discover these details on their own.

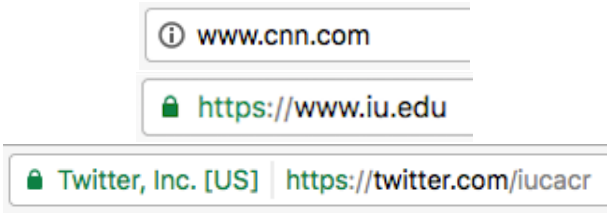

Figure 1: Security visual decorators in a browser.

3.1 X.509 Certificate

Just as a birth certificate provides a person an authentic document stating their birth date and place, an *X.509 certificate* is intended to authenticate an entity on the Web. And just like a birth certificate, an X.509 certificate is issued by some trusted authority.

The contents of an X.509 certificate (currently at version 3) are defined in RFC 5280. We list just a few of its fields:

Serial Number: Used to uniquely identify the certificate.

[1]Perhaps more accurately, *rediscovery* and made public for the first time.

[2]http://amturing.acm.org/

[3]Using the Transport Layer Security (TLS) protocol.

Subject: The entity associated with the public key. (e.g., www.iu.edu)
Signature Algorithm: The algorithm used to create the signature. (e.g., SHA-256 with RSA Encryption)
Signature: The digital signature to verify that it came from the issuer. (string of bytes)
Issuer: The entity that verified the information and issued the certificate. (e.g., Internet2/InCommon)
Valid-From: The start date of the certificate's validity. (e.g., November 6, 2014)
Valid-To: The end date of the certificate's validity. (e.g., November 6, 2017)
Public Key: Public key associated with this certificate. (string of bytes)
Issuing Distribution Point: CRL (a URI)
Authority Information Access: OCSP Responder (a URI)
Fingerprint: A hash of the certificate.

We emphasize the fact that a certificate is not some mysterious, hidden entity from a Web user. This is in keeping with the goal of providing an open, secure Internet that Diffie, Hellman, and others had in mind from the beginning. If a user wants to investigate the detailed contents of the certificates associated with a secure HTTPS web site, simply click the padlock decorator to the left of the address bar and then a *Details* and/or *View Certificate* button that subsequently appear. Figures 3 and 4 show some of this information (in the Chrome browser). Clicking on any one of the Certificate Authorities (discussed next) (Root, Intermediate, or Leaf) will reveal details specific to its certificate.

3.2 Certificate Authority

A certificate authority (CA) is a trusted, third-party entity who issues and, when necessary, revokes certificates. The CA's digital signature becomes part of the certificate itself so that it can be verified. Quoting from RFC 5280:

> CAs are responsible for indicating the revocation status of the certificates that they issue. Revocation status information may be provided using the Online Certificate Status Protocol (OCSP) [RFC2560], certificate revocation lists (CRLs), or some other mechanism. In general, when revocation status information is provided using CRLs, the CA is also the CRL issuer. However, a CA may delegate the responsibility for issuing CRLs to a different entity.

Not surprisingly, most CAs charge a fee, e.g., a few hundred dollars per year, to issue certificates and provide the necessary services associated with them. However, there is currently at least one (automated) CA offering certificates and services for free[24].

The large number of HTTPS Web sites necessitates a hierarchical structure of CAs. The idea of having a single CA be responsible for all certificates is not only technically infeasible, it is a really bad idea from a security perspective. Single points of failure are never good. Therefore, we have a hierarchy that consists of *root* CAs, *intermediate* CAs, and *leaf* CAs, as depicted in Figure 2. Root CAs are the most trusted and (hopefully) the most secure. Root CAs issue certificates to intermediate CAs and intermediate CAs issue certificates to other intermediate CAs or to leaf CAs, the end-entity in the chain. Thus, there is a unique *path*, or *chain*, from a leaf CA to a root CA. As we discuss in Section 4, when a browser validates the (leaf) certificate associated with a Web site, it must validate all certificates in that chain.

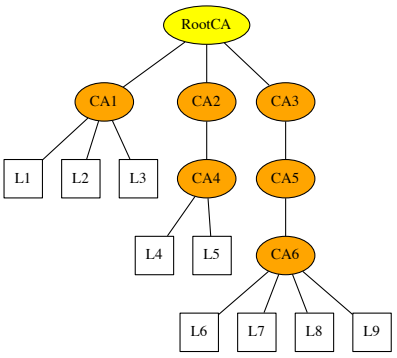

Figure 2: Hierarchical CA model

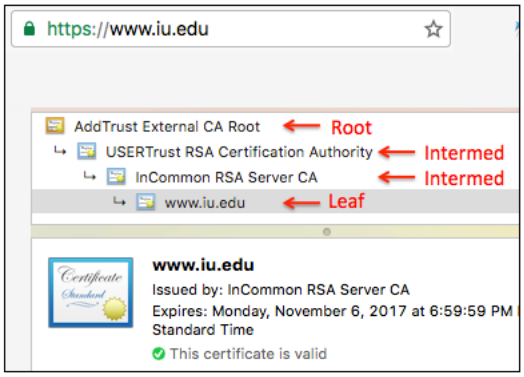

Figure 3: A certificate and its chain of CAs.

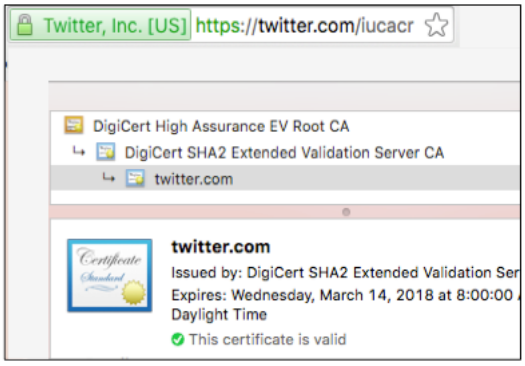

Figure 4: An EV certificate.

For those who would like to see more details of the information exchanged during the secure (TLS) handshake, including the (encoded) certificates, the free OpenSSL soft-

ware allows this. For example, running the following command[4] from a terminal window will return information related to the leaf certificate of the specified Web server:

```
$ openssl s_client -connect www.iu.edu:443
```

Appending a `-showcerts` argument to the above command will output all certificates in the chain.

4. CERTIFICATE VALIDATION

When a user visits a HTTPS Web site and the TLS handshake occurs between a browser and Web server, the browser receives a leaf certificate and must validate it. However, since there is a path of certificates connected to that leaf, the process is actually multi-step. RFC 5280 describes the conditions that need to be met for *certificate path validation*:

1. for all x in 1, ..., n-1, the *subject* of certificate x is the *issuer* of certificate x+1;
2. certificate 1 is issued by the trust (root) anchor;
3. certificate n is the certificate to be validated (i.e., the target certificate); and
4. for all x in 1, ..., n, the certificate was valid at the time in question.

Part of the process of determining whether an individual certificate is valid includes 1) verifying the current date is within its *Valid-From/To* dates, and 2) checking that it has not been revoked[5]. It is this latter step, checking for certificate revocation, that we explore more fully in this section by examining various implementations.

Liu *et al.*[23] provide a review of certificate revocation implementations and analyze the myriad combinations of browsers and operating systems, showing how each may process the revocation status differently. We discuss some of the same implementations here, albeit for a more general audience.

4.1 Certificate Revocation Lists

Certificate Revocation Lists (CRLs) are just that - lists of certificates that have been revoked. An entry in a CRL contains a certificate's (unique) serial number and the date it was revoked. The entire CRL file is digitally signed by the CA that maintains it. RFC 5280 states that either a CA or an entity authorized by the CA can issue CRLs and lists eight reasons why a certificate might be revoked, including two of the more egregious: the certificate's private key was compromised or, worse yet, a CA was compromised.

When a browser receives a certificate, part of the validation process is to check if the certificate has been revoked. Assuming the certificate contains a CRL *issuing distribution point*, the browser can download the entire CRL and check to see if it contains this certificate's serial number. If it does not, then the browser would check for the next certificate in the chain. If it reaches the RootCA (trust anchor) and has not found it in a CRL, the browser will verify the RootCA is in its pre-loaded list of trusted root CAs.

There are some obvious challenges and overhead costs associated with CRLs. Maintenance, i.e., keeping it up to date and secure, is primarily a burden of the CAs. The browsers and client computers incur the costs associated with processing the CRL, i.e. verifying it is valid and searching it for certificates. We will discuss costs more in Section 5.

4.2 Online Certificate Status Protocol

The Online Certificate Status Protocol (OCSP)[32] provides an alternative to the CRL. Rather than have a browser download an entire CRL file, OCSP offers a lighter weight solution. The browser can now just send the certificate to an OCSP *responder* and receive a status of good, revoked, or unknown.

Not surprisingly, there are trade-offs with both CRLs and OCSP. Although OCSP suggests less data transmission, it is possible an OCSP responder would be down. This would mean a browser could not determine the revocation status of a certificate; whereas, if the browser had downloaded an entire CRL, the revocation check might be a local operation.

4.3 OCSP Stapling

An extension of the OCSP concept is actually an extension to the TLS protocol[14]. OCSP Stapling[29] will let a Web server cache OCSP results (good, revoked, or unknown) and send that result to the browser during the TLS handshake. This would eliminate the need for the browser to contact an OCSP responder.

Once again there are trade-offs. Not all Web servers support OCSP Stapling. The SSL Pulse[6] reports that, as of September 2016, only about 25% of the "most popular" Web sites support OCSP Stapling. And the analysis in [23] suggests a much lower percentage by less popular Web sites.

4.4 Public Key Pinning

Public Key Pinning[25], commonly referred to as HTTP Public Key Pinning (HPKP), is an effort to help circumvent untrusted certificates from compromised CAs. The HPKP protocol[9] defines a new HTTP header that lets Web servers tell browsers to remember, or "pin", their public keys over a period of time. During that time, user agents (UAs) will require that the host presents a certificate chain including at least one Subject Public Key Info structure whose fingerprint matches one of the pinned fingerprints for that host.

HPKP was initiated at Google and is supported by their Chrome browser (>= version 46), as well as recent versions of Firefox; however, other leading browsers do not currently support it. By reducing the number of trusted authorities who can authenticate the domain during the lifetime of the pin, pinning may reduce the incidence of man-in-the-middle attacks due to compromised CAs.

Results from the first comprehensive survey of HPKP are reported in [19]. One observation made there is that HPKP will not resolve the security issues surrounding mixed content, i.e., when an HTTPS page loads resources from an HTTP origin.

4.5 Certificate Transparency

As its name suggests, Certificate Transparency (CT)[1] is an attempt to introduce greater transparency into the traditional CA-issued certificate system. It is another Google-backed project and the protocol is still considered experimental[21]. Laurie and Doctorow have made a succinct argument for its adoption[20] and provide cases of compromised CAs for justification. The basic idea is that when

[4]443 is the default port for HTTPS.

[5]There are other important steps, e.g., verifying signatures, that we skip over here.

[6]https://www.trustworthyinternet.org/ssl-pulse/

a CA issues a certificate, it sends a copy to a log. This log server will be append-only, public (hence, transparent), secure, monitored and audited. The security of the logs is due, in part, to something called Merkle[7] trees[28]. Browsers could be pre-loaded with a list of these log servers, just as they are pre-loaded with a list of trusted root CAs for CRLs.

The Let's Encrypt[8] CA referenced in Section 3.2 uses CT.

4.6 CRLSets

The Google Chrome browser has yet another approach for checking for revoked certificates. By default, Chrome does not perform a typical online, e.g., OCSP, revocation check for certificates. Rather, Google crawls CAs and gathers serial numbers of revoked certificates. The Chrome browser is then updated with these *CRLSets* at least once a day and revocation checking becomes a "local" operation against the CRLSet. This is an instance of the browser developer (Google) shouldering the major cost of the certificate revocation check.

5. MODELING COST

Jeannette Wing has been a leader in developing and promoting the concept of *Computational Thinking*[40][39]. One of the characteristics that helps define this skill set is "conceptualizing, not programming". She offers the following:

> Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable.

In this section, *our goal is to help the (student) reader conceptualize how one might model costs associated with the various certificate validation implementations* discussed in the previous section. Clearly, modeling these costs is a complex problem. As we have discussed, it involves a number of different *actors*: browsers (clients), Web servers, CAs, CRL servers, OCSP responders, etc. The network(s) could even be considered actors. And the actors will have different states and different behavioral parameters, e.g., X.509 certificates, CRLs, CPU speeds, number of cores, memory, bandwidth, latency, etc. One could therefore imagine a cost function that would involve many parameters:

$$Cost = F(x_1, x_2, ..., x_N)$$

So how should we begin? Following Wing's advice, we decompose the problem and try to model just the relevant aspects to make it more tractable. For example, let's begin by just considering CRL and OCSP models and the cost of moving the data associated with checking for certificate revocation over the network. Therefore, we define two functions, each dependent on a single parameter, the size of the data that is transferred:

$$Cost_{CRL} = F(revSize)$$
$$Cost_{OCSP} = F(revSize)$$

How does one determine meaningful estimates for parameters? There are essentially two choices: perform your own experiments to gather data or use data from published experiments. Choosing the latter, we look at the experiments performed in [23] and see that the size of CRLs can vary considerably, from tens of kilobytes to tens of megabytes. Obviously, the size of CRLs will spike if, for example, a CA has been compromised and all the certificates it has issued need to be revoked. The size of data being transmitted for OCSP is going to be much smaller.

One might think latency, the amount of time it takes for a data packet to travel between two endpoints (e.g., a browser and a Web server), would be an important parameter for determining cost. Indeed, there is considerable variability in network latency, as anyone can see using the *ping* command from a desktop terminal window. This is demonstrated by pinging a "local" Web server versus one half way around the world:

```
$ ping www.iu.edu
PING www.iu.edu : 56 data bytes
: icmp_seq=0 ttl=61 time=5.004 ms
: icmp_seq=1 ttl=61 time=4.354 ms
: icmp_seq=2 ttl=61 time=16.427 ms

$ ping www.tsinghua.edu.cn
PING www.d.tsinghua.edu.cn: 56 data bytes
: icmp_seq=0 ttl=233 time=306.366 ms
: icmp_seq=1 ttl=233 time=329.849 ms
: icmp_seq=2 ttl=233 time=282.476 ms
```

But is latency really that relevant for the problem of certificate validation? This is the type of question a modeler needs to ask (and answer). Another important question to keep in mind for modeling the cost of certificate validation is: the cost *for whom*? After all, we have multiple actors, as already mentioned above. For example, the cost that a browser incurs will likely be very different from the cost incurred by a Web server, or OCSP responder, or any other actor in our system.

One important feature the ping demonstration does reveal is *noise*. Note the varying times a data packet takes to the same destination. There will always be noise, or randomness, in measuring and modeling complex systems. Therefore, in the generic *Cost* equation above, one of the x_i parameters would, in fact, be noise. Another fairly obvious parameter would be the depth of the certificate path that was discussed in Section 3.2. The revised Cost function then becomes:

$$Cost = F(revSize, caDepth, noise)$$

6. SIMULATION RESULTS

After creating conceptual models to calculate the costs of certificate validation, one needs to transform them into computer code. And when dealing with complex systems, e.g., that involves multiple actors, parameters, and randomness, the resulting code will be a system *simulation*, rather than a simple, straightforward calculation.

Just as we had two choices for obtaining data for input parameters, here too, we have two choices: write the simulation code yourself or adopt existing code. For this paper, we adopt existing code (and tailor it for our needs). The Application-Sensitive Access Control (ASAC) analysis

[7]Ralph Merkle was, along with Diffie and Hellman, one of the early leaders in PKC. He was a student of Hellman.
[8]https://letsencrypt.org/certificates

framework has been used to simulate a variety of access control models[12][13][11]. Since PKI can be considered an access control mechanism (rf. statement from RFC 5280, Section 3), the ASAC framework seems appropriate. However, its use in other models is much more formal and exhaustive than we will attempt here. Our use of the ASAC software will be simple and suggestive, in the hope that students can learn from the modeling and simulation and expand upon it.

An ASAC simulation allows a modeler/programmer to: define actors, define actors' actions (commands and queries), define costs (measures), perform multiple runs on multiple models, and automatically plot results. The simulation software is written in Java and plotting uses the open source matplotlib[15] (Python) package. By default, all pairwise plots of measures will be generated.

Figures 5 and 6 show pairwise plots for two models, CRL and OCSP, and two measures, CA depth and certificate revocation response size. The model associated with Figure 6 had more noise associated with the response size.

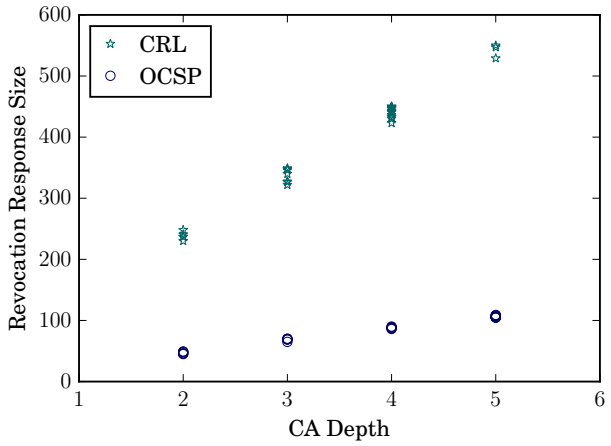

Figure 5: CRL,OCSP: RevSize,CA depth,low noise

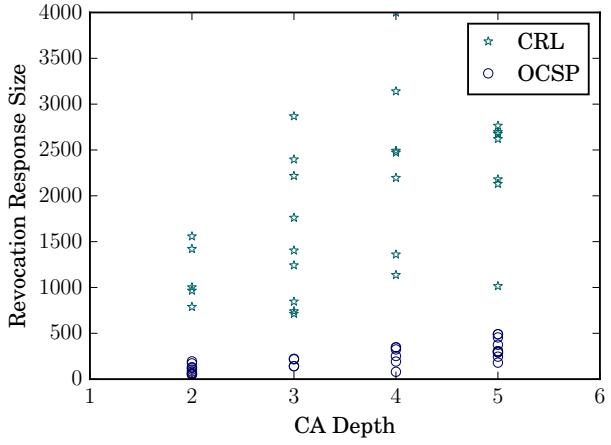

Figure 6: CRL,OCSP: RevSize,CA depth,high noise

7. RELATED WORK

The simulation results presented here could possibly be replicated using other software packages[9]. Any extensible, discrete event simulation engine that models a state machine of generic actors would be a good starting point. The ns-3 software[30], for example, has been used to evaluate the scalability of PKI [38] for models of the *Smart Grid*. In another project, SSFnet[6] has been used to evaluate a very large scale model of certification path discovery[42].

8. CRYPTO AND NUMBER THEORY

We have primarily discussed PKI and certificate validation in this paper. In this last section, we depart from this focus and discuss public key cryptography (PKC). More specifically, we discuss its connection to mathematics, primarily number theory. PKC is responsible for the actual encryption/decryption (cryptography) of the data that is transmitted between a browser and the servers with whom it communicates.

As mentioned in Section 2, Diffie and Hellman[7] discussed one-way functions as a possible solution to PKC. They used matrix inversion as an illustrative, but non-practical example. The RSA paper[31] then revealed a practical one-way function that would become responsible for the majority of PKC used by the Web. In hindsight, this one-way function seems rather obvious: larger integer factorization. You may recall from a high school math class that integer factorization leads us to the topic of prime numbers.

By definition, a *prime number* is a positive integer > 1 that cannot be factored. For example, 7 is prime; 15 is not prime since it can be factored as $3 * 5$. The *Fundamental Theorem of Arithmetic* states that every integer > 1 is either prime or factors uniquely as a product of primes. For this reason, primes have been called the atomic elements of integers.

It is enlightening to experiment with computational number theory and we encourage readers to do so. Mathematica is a full-featured commercial software package (and free if used on a Raspberry Pi[10] computer). Another popular, open source package is SageMath. There are also freely available books that provide an introduction to computational number theory[35][2].

In the context of PKI, PKC, and RSA, an obvious starting point would be to explore the topic of integer factorization. We demonstrate with SageMath and some relatively small integers:

```
sage: factor(1234)
2 * 617
sage: factor(45678)
2 * 3 * 23 * 331
sage: factor(56789)
109 * 521
sage: factor(1111199999)
7^2 * 13 * 41 * 157 * 271
```

The factors for these are returned essentially instantaneously. For much larger integers, as we will see below, this is not the case.

[9]It is, in fact, good practice to demonstrate reproducible results (at least qualitatively) using multiple software packages.

[10]http://www.wolfram.com/raspberry-pi/

It is interesting to think that prime numbers were once considered just an esoteric topic of number theory, with no known "real world" applications. G.H. Hardy, a renowned pure mathematician once declared[8] that mathematicians could rejoice knowing that number theory is so remote from human activities that it would be kept "gentle and clean". Since he lived in the mid-20th century, he did not foresee the Internet, its need for privacy, and a solution that would involve one-way functions, integer factorization, and prime numbers. Of course the RSA algorithm, with its use of large integer factorization is not a unique one-way function. As a teenager, Sarah Flannery[10] became something of a sensation when she proposed an alternative in 1999. While her suggested algorithm did not ultimately withstand rigorous testing, it was a testament to those interested in exploring mathematical alternatives to the problem.

Factoring large integers has a fascinating history. In 1903, Frank Cole revealed the two prime factors of the 21-digit number $2^{67} - 1$. He had spent "3 years of Sundays" to discover this factorization [3]. Today, thanks to computers and advances in computational number theory, we can factor this same number in milliseconds:

```
sage: time factor(2^67 - 1)
CPU times: user 6.5 ms
```

It may come as a surprise that determining whether or not a number is prime versus actually finding the factors of the number are quite different. The latter is a *much* more difficult problem. Consider factoring the following 40-digit number (using SageMath):

```
sage: c=2630492240413883318777134293253671517529
sage: time is_prime(c)
CPU times: user 80 µs
False
sage: time factor(c)
CPU times: user 93.8 ms
48112959837082048697 * 54673257461630679457
```

Obviously this is not conclusive evidence, but assuming SageMath is using reasonably state-of-the-art algorithms, then it takes roughly a thousand times longer to factor this particular number (millisecs) than to determine if it is prime (microsecs).

The next two examples are simply intended to demonstrate how difficult (time-consuming) it is for computers to factor large integers. Given 2 N-digit primes, we multiply them together and then ask Sage to factor the product. When $N = 30$, it takes only 10 seconds; however, when $N = 40$, it takes 13 minutes. Imagine if N is much larger. We encourage readers to experiment for themselves. SageMath also provides a free cloud service[11].

```
# time to factor semiprime (of two 30-digit primes)
sage: p30=282174488599599500573849980909
sage: q30=115756986668303657898962467957
sage: pq=p30*q30
sage: time factor(pq)
CPU times: user 10 s
115756986668303657898962467957 *
282174488599599500573849980909
```

[11]https://cloud.sagemath.com

```
# factoring two 40-digit primes
sage: p40=2425967623052370772757633156976982469681
sage: q40=1451730470513778492236629598992166035067
sage: pq=p40*q40
sage: time factor(pq)
CPU times: user 13min 22s
1451730470513778492236629598992166035067 *
2425967623052370772757633156976982469681
```

A natural question to ask is how long integer factorization will continue to be the dominant one-way function used for PKC. There are at least two, yet to be realized breakthroughs that may affect this (negatively). One is something called quantum computation[34]. Another, much more fundamental to mathematics, is called the Riemann Hypothesis[26] and has been named a Millennium Prize Problem[16], with a one million dollar prize for its proof. We encourage readers to learn more about these exciting topics. Finally, it should be noted that integer factorization is not the only option available for PKC; elliptic curve cryptography[36][27] is another popular choice.

9. CONCLUDING DISCUSSION

We have reviewed some of the key ideas and interesting history behind PKI and PKC. X.509 certificates play a fundamental role in PKI and we have demonstrated how anyone using a Web browser can examine detailed information contained in a certificate. The crucial step of certificate validation, including the process of checking for revoked certificates, is an evolving process. We have discussed multiple implementations and some of their trade-offs. We discussed briefly the challenge of modeling and provided a simple model for calculating costs of certificate validation. The modeling process led to simulation and we demonstrated using the ASAC software framework. There is much more that can be explored in modeling/simulating the PKI certificate validation process and we hope we have sparked students' interest.

Various stakeholders in the PKI landscape have different incentives to improve it (or not). CAs, for example, would probably prefer *not* to spend money on more modern computers acting as OCSP responders. Web browser developers may not want to implement a new, experimental certificate validation protocol if its fate is unknown. Other stakeholders likely have other (dis)incentives. With improvements in PKI modeling, simulation, and analysis of results, perhaps we can provide more evidence to support future decisions and help make the Web more secure for all of us.

10. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation through the awards 1228668 and 1228697. We thank and acknowledge contributions from students at Indiana University: Rohan Mathure, Veer Singh, and Rahul Sinha. We also thank Jim Basney, NCSA/UIUC, for helpful comments.

11. REFERENCES

[1] Certificate transparency. https://www.certificate-transparency.org/. Accessed: 2016-8-11.

[2] Discovering the art of mathematics: Number theory. http://artofmathematics.org/books/number-theory. Accessed: 2016-08-06.

[3] One 'long' multiplication problem... revisited. https://musingsonmath.com/tag/frank-cole/, 31 Oct. 2012.

[4] K. Conger. Chrome is helping kill HTTP. https://techcrunch.com/2016/09/08/chrome-is-helping-kill-http/, 8 Sept. 2016.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, May 2008.

[6] J. Cowie, A. Ogielski, and D. Nicol. The SSFNet network simulator. *Software on-line: http://www.ssfnet.org/homePage.html*, 2002.

[7] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, Nov. 1976.

[8] M. du Sautoy. *The Music of the Primes: Searching to Solve the Greatest Mystery in Mathematics*. HarperCollins, 2003.

[9] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP. RFC 7469, 2015.

[10] S. Flannery. *In Code: A Mathematical Journey*. Workman Pub., 2000.

[11] W. C. Garrison, A. J. Lee, and T. L. Hinrichs. The need for application-aware access control evaluation. In *Proceedings of the 2012 Workshop on New Security Paradigms*, NSPW '12, pages 115–126, New York, NY, USA, 2012. ACM.

[12] W. C. Garrison, A. J. Lee, and T. L. Hinrichs. An actor-based, application-aware access control evaluation framework. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies*, SACMAT '14, pages 199–210, New York, NY, USA, 2014. ACM.

[13] W. C. Garrison, A. J. Lee, and S. Myers. On the practicality of cryptographically enforcing dynamic access control policies in the cloud. In *2016 IEEE Symposium on Security and Privacy*, May 2016.

[14] Huawei. Transport layer security (tls) extensions: Extension definitions. RFC 6066, Jan. 2011.

[15] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[16] J. Carlson, A. Jaffe, and A. Wiles, editor. *The Millennium Prize Problems - Clay Mathematics Institute*. American Mathematical Society and Clay Mathematics Institute, 2006.

[17] J. Jonsson and B. Kaliski. Public-key cryptography standards (pkcs) 1: Rsa cryptography specifications version 2.1. Technical report, RFC Editor, Feb. 2003.

[18] L. M. Kohnfelder. *Towards a Practical Public-key Cryptosystem*. PhD thesis, MIT, May 1978.

[19] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Proceedings 2015 Network and Distributed System Security Symposium*, Reston, VA. Internet Society.

[20] B. Laurie and C. Doctorow. Computing: Secure the internet. *Nature*, 491(7424):325–326, 11 2012.

[21] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, June 2013.

[22] S. Levy. *Crypto: How the Code Rebels Beat the Government–Saving Privacy in the Digital Age*. Penguin USA, New York, NY, USA, 2001.

[23] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson. An End-to-End measurement of certificate revocation in the web's PKI. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 183–196. ACM, 28 Oct. 2015.

[24] N. Lomas. Let's encrypt free HTTPS certification push exits beta. https://techcrunch.com/2016/04/12/lets-encrypt-free-https-certification-push-exits-beta/, 12 Apr. 2016.

[25] R. Love. Everything you need to know about HTTP public key pinning (HPKP). http://blog.rlove.org/2015/01/public-key-pinning-hpkp.html, 20 Jan. 2015.

[26] B. Mazur and W. Stein. *Prime Numbers and the Riemann Hypothesis*. Cambridge University Press, Cambridge, 2016.

[27] D. McGrew, K. Igoe, and M. Salter. Fundamental elliptic curve cryptography algorithms. Technical report, RFC Editor, Feb. 2011.

[28] R. C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology — CRYPTO '89 Proceedings*, Lecture Notes in Computer Science, pages 218–238. Springer New York, 20 Aug. 1989.

[29] Y. Pettersen. The transport layer security (TLS) multiple certificate status request extension. RFC 6961, June 2013.

[30] G. F. Riley and T. R. Henderson. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*, pages 15–34. Springer, 2010.

[31] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, Feb. 1978.

[32] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - OCSP. RFC 6960, June 2013.

[33] T. Scott. HTTPS-Everywhere for government. https://www.whitehouse.gov/blog/2015/06/08/https-everywhere-government, 8 June 2015.

[34] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. ieeexplore.ieee.org, Nov. 1994.

[35] W. Stein. *Elementary Number Theory: Primes, Congruences, and Secrets: A Computational Approach*. Undergraduate Texts in Mathematics. Springer New York, 2008.

[36] S. Turner, D. Brown, K. Yiu, R. Housley, and T. Polk. Elliptic curve cryptography subject public key information. Technical report, RFC Editor, Mar. 2009.

[37] P. Vryonis. Explaining public-key cryptography to non-geeks. https://medium.com/@vrypan/

explaining-public-key-cryptography-to-non-geeks-f0994b3c2d5#.zbpgdupnz, 27 Aug. 2013.

[38] T. L. Ward. Grid Cryptographic Simulation: A Simulator to Evaluate the Scalability of the X.509 Standard in the Smart Grid. Technical Report TR2013-742, Dartmouth College, Computer Science, Hanover, NH, September 2013.

[39] J. M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, Mar. 2006.

[40] J. M. Wing. Computational thinking, 10 years later. https://www.microsoft.com/en-us/research/computational-thinking-10-years-later/, 23 Mar. 2016.

[41] L. Zhang, D. Choffnes, D. Levin, T. Dumitras, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 489–502, New York, NY, USA, 2014. ACM.

[42] M. Zhao and S. W. Smith. Modeling and evaluation of certification path discovery in the emerging global PKI. In *Public Key Infrastructure*, Lecture Notes in Computer Science, pages 16–30. Springer Berlin Heidelberg, 19 June 2006.