Master Thesis

STUNT
A Simple, Transparent, User-centered Network of Trust

Klaus Potzmader

klaus.potzmader@student.tugraz.at

August 2013

Advisors

Bloem, Roderick Paul, Univ.-Prof. M.Sc. Ph.D.

Hein, Daniel, Dipl.-Ing.

Graz University of Technology

Institute for Applied Information Processing and Communications

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..
date (signature)

Acknowledgements

First of all, I want to thank Daniel Hein and Johannes Winter, for their feedback and patience, as well as their support in developing this new idea, which I just threw at them one day, right out of the blue. We had many interesting meetings, of which both this work and myself benefited thoroughly. Furthermore, I want to thank Roderick Bloem for accepting the initial proposal and examining this thesis. My gratitude also goes to the IAIK as a whole, for the excellent teaching and for providing me with the infrastructure that was needed to build the lab environment.

My thanks also go to my fellow students, with whom I had several beneficial and interesting discussions about the problem. Furthermore, we were a great team throughout the studies, always motivating and learning from each other, which was a great experience for me that i do not want to miss. Moreover, we had a great time aside from the studies.

Next, I thank my parents and family for believing in me and supporting me during the whole studies. I might have not seen you as much as I wanted in that time, but I certainly recognized your support and you were there when I needed you. Last but certainly not least, I want to thank my girlfriend Cornelia for all the understanding and backing, especially in times where I was completely focused on work.

Looking back, I am grateful that it was possible for me to attend Graz University of Technology, which would not have been possible and such a great, powerful experience without the people mentioned above.

"I was told when I grew up I could be anything I wanted: a fireman, a policeman, a doctor — even president, it seemed. And for the first time in the history of mankind, something new, called an astronaut. But like so many kids brought up on a steady diet of westerns, I always wanted to be the avenging cowboy hero — that lone voice in the wilderness, fighting corruption and evil wherever I fount it, and standing for freedom, truth, and justice. And in my heart of hearts I still track the remnants of that dream wherever I go, in my endless ride into the setting sun."

– Bill Hicks

Abstract

Secure end-to-end communication requires assurance about endpoint authenticity. Authenticating an endpoint in large networks, i.e., assuring that the other communication party is indeed who is being claimed, is a non-trivial task. Public-key cryptography alleviated the key exchange problem, allowing two parties to communicate in an encrypted way without having to exchange a common, secret key beforehand. However, there is no binding between the public key presented by a host and the person operating this computer. Verifying that the key owner is indeed the claimed person is especially hard to assess in an Internet context, where communicating parties often neither know nor see each other personally.

Currently, the adopted solution addressing this binding problem is to rely on trusted third parties, who are supposed to ensure a certain host's authenticity. Trusted third parties verify the identity of an operator, and subsequently vouch for the claimed binding between operator and key, if successful. Unfortunately, recent incidents at renowned trusted third parties, as well as long standing problems, indicate a need for alternative solutions, not involving external, all-trusted entities.

We propose a Simple, Transparent, User-centered Network of Trust (STUNT), a new effort to address this problem from a different angle. It helps users to assess a host's authenticity by its trust relationships with other hosts. Hosts operated by service providers have to establish mutual trust relationships with other service providers to appear trustworthy to a user. Trust relationships express the confidence of a service operator, that another host is indeed operated by whom it claims to be. These trust relationships are defined to be mutual, to encourage careful decision-making upon establishment. Carelessly engaging in a trust relationship might backfire at the initiator, since the trusting host will become an integral part of the initiator's own reputation. Furthermore, these trust relationships are both limited and expensive, to put even more emphasis on careful trust decisions made by service operators. Clients are able to verify these trust relationships on the fly by cryptographic means. The verified trust relationships are presented to users as a graph, to assist them in assessing the authenticity of the host. Therefore, a host is verified by examining valuable and thus meaningful references. Verifying an unknown entity by means of recommendations of others is an old, well-known concept, and thus considered appropriate for intuitive, yet informed decision-making. Ultimately, the trust decision rests with the user, leading to an individual, self-maintained trust base. Such a self-created, personal trust base is considered highly desirable, especially when compared to the currently used lists of trusted roots, delivered with operating systems, browsers and runtime environments in a predefined way.

We believe that people, given the right tools and information, are able to decide on a host's authenticity. STUNT describes a possible technical concept, supporting users in making informed decisions, while still minimizing the impact on usability in terms of the frequency of interruptions.

Keywords: *network, Internet, SSL, authenticity, trust, reputation, web of trust*

Kurzfassung

Sichere Ende-zu-Ende Verbindungen erfordern die Sicherstellung der Authentizität der involvierten Endpunkte. Einen Endpunkt in großen Netzwerken zu authentifizieren, also sicherzustellen, dass dieser tatsächlich die behauptete Entität ist, ist schwer zu bewerkstelligen. Public-Key Kryptographie erleichtert das Problem des Schlüssel-Austauschs für verschlüsselte Kommunikation, indem es damit nicht mehr notwendig ist, einen gemeinsamen, geheimen Schlüssel auszutauschen. Allerdings gibt es keine garantierte Bindung zwischen einem öffentlichen Schlüssel und der benutzenden Person. Festzustellen, dass der Schlüsselbesitzer also tatsächlich jene behauptete Person ist, ist speziell in Internet-Szenarien sehr schwer, weil sich die kommunizierenden Parteien in diesem Kontext oftmals weder persönlich kennen noch jemals trafen.

Der derzeit verwendete Ansatz um diesem Problem beizukommen, ist, sich auf sogenannte Trusted Third Parties zu verlassen, welche für die Authentizität eines Hosts oder einer Person bürgen. Zahlreiche in jüngster Zeit bekannt gewordene Probleme mit diesen Trusted Third Parties, sowie lange bekannte Probleme, zeigen allerdings auf, dass neue Lösungen angebracht sind.

Ein Simples, Transparentes, User-zentries Netzwerk von Trust (STUNT), ist ein neuartiger Ansatz zur Abschwächung des Problems der Feststellung der Authentizität von Computern oder Personen. Es wird dabei eine andere, Benutzer-zentrierte Sichtweise verfolgt, bei der Benutzer selbst deren Vertrauen in einen Dienstbetreiber definieren. Diese Vertrauensevaluierung erfolgt via Plausibilitätsprüfung von Vertrauensbeziehungen mit anderen Dienstbetreibern. Diese Vertrauensbeziehungen sind als beidseitig definiert, und für Dienstbetreiber essenziell, um vertrauenswürdig zu erscheinen. Beidseitig deshalb, damit unüberlegte Vertrauensentscheidungen, welche dem gesamten System schaden könnten, primär dem Entscheidungsträger selbst schaden, was zu vorsichtiger Handhabung führen soll. Desweiteren sind diese Vertrauensbeziehungen pro Dienst in ihrer Anzahl begrenzt und teuer im Aufbau. STUNT forciert damit überlegte Entscheidungen seitens der Dienstbetreiber, was die Auswahl der Vertrauensbeziehungen anbelangt. Clients können die Vertrauensbeziehungen eines Dienstes innerhalb kürzester Zeit mittels kryptografischer Verfahren überprüfen. Nach abgeschlossener Überprüfung werden diese Vertrauensbeziehungen dem Benutzer als Graph präsentiert, um diesem bei der Authentizitätsfeststellung zu helfen. Ein Dienst wird daher mittels wertvoller und damit bedeutsamer Referenzen authentisiert. Sich mittels Empfehlungen einen Eindruck über etwas bislang Unbekanntes zu verschaffen ist ein altes, bekanntes Konzept. Es scheint daher in diesem Kontext brauchbar für das Treffen von intuitiven, informierten Entscheidungen. Die Vertrauensentscheidung obliegt dem Benutzer, was zu persönlichen, selbstgewarteten Vertrauensfundamenten führt. Diese selbsterstellten Vertrauensfundamente sind wünschenswert, speziell wenn man diese mit dem herkömmlichen Weg der vordefinierten, vertrauenswürdigen Hosts vergleicht, welche mit Browsern, Betriebssystemen oder Laufzeitumgebungen mitgeliefert werden.

Wir glauben, dass Benutzer, richtiges Werkzeug und relevante Informationen vorausgesetzt, die Authentizität von Dienstbetreibern selbst feststellen können. STUNT beschreibt ein mögliches technisches Konzept um dieses Verhalten herbeizurufen. STUNT bleibt dennoch benutzbar, indem es Störungen des Benutzers durch oftmaliges Unterbrechen des Arbeitsflusses minimiert.

Schlüsselwörter: *Netzwerk, Internet, SSL, Authentizität, Vertrauen, Reputation, Web of Trust*

Contents

List of Figures

List of Tables

List of Acronyms

CA	Certificate Authority ... 1
CIA	Confidentiality, Integrity, Availability ... 11
CPU	Central Processing Unit ... 55
CRL	Certificate Revocation List ... 20
DANE	DNS-based Authentication of Named Entities ... 30
DBMS	Database Management System ... 51
DHCP	Dynamic Host Configuration Protocol ... 55
DN	Distinguished Name ... 18
DoS	Denial-of-Service ... 39
DV	Domain Verification ... 17
DNS	Domain Name System ... 29
DNSSEC	Domain Name System Security Extensions ... 15
ECIES	Elliptic-Curve Integrated Encryption Scheme ... 11
ECU	Amazon EC2 Compute Unit ... 46
EFF	Electronic Frontier Foundation ... 2
EV	Extended Verification ... 17
GPU	Graphics Processing Unit ... 46
GUI	Graphical User Interface ... 52
HTTP	HyperText Transfer Protocol ... vii
HTTPS	HyperText Transfer Protocol (HTTP) Secure ... 2
IETF	Internet Engineering Task Force ... 21
IP	Internet Protocol ... 29
JDBC	Java Database Connectivity ... 51
LDAP	Lightweight Directory Access Protocol ... 18
MAC	Message Authentication Code ... 11
MD5	Message Digest 5 ... 4
OCSP	Online Certificate Status Protocol ... 20
OID	Object Identifier ... 18
PCR	Platform Configuration Register ... 14
PGP	Pretty Good Privacy ... 15
PKI	Public Key Infrastructure ... 23
PKIX	A family of standards and RFCs for public-key infrastructures, including X.509 and the RFCs 5280 and 2560

Chapter 1

Introduction

"The world has become so complex that the idea of a power in which everything comes together and can be controlled in a centralized way is now erroneous."
– Ulrich Beck

Communication is only to be labeled secure if both parties are ensured that they are indeed dealing with each other. Encryption and message integrity assurance is essential for secure data exchange, but pointless if carried out with the wrong person. Therefore, authenticity is an important, integral part of secure communication. Nowadays, with more than 60% of all people in the western world having Internet access, and more than 34% overall, it is hardly assumable that all communicating parties ever met, or will meet, in person [Miniwatts Marketing Group, 2013]. Thus, a personal key exchange as a trusted base for future secure communication is plainly unrealistic in most cases.

A Simple, Transparent, User-centered Network of Trust (STUNT) is an attempt to increase confidence in the authenticity of communication endpoints. In scenarios where the involved endpoints have never met before, there is no absolute assurance that the participants are indeed who they claim to be. Nevertheless, STUNT is all about preparing informed decision-making for users. It aims to make this factor of uncertainty as explicit as possible, but delegates the ultimate trustworthiness decision to the user. Therefore, it strongly differs from the currently adopted approach of Public Key Infrastructure X.509 (PKIX) [IETF PKIX Working Group, 2013], which uses Trusted Third Parties (TTPs) to take this decision away from users.

1.1 Present Situation

Currently, the PKIX solution, adopted by both operators and major software packages, is to employ Certificate Authorities (CAs) to vouch for the authenticity of other hosts. CAs vouch for the authenticity of another entity in a hierarchical manner, with the most influental CAs, called root CAs, residing at the top of such a hierarchy tree. These root CAs vouch for the authenticity of either intermediate CAs or endpoints. Root CAs are pre-installed in browser software or the operating system and denote ultimately trusted entities, capable of deciding about another's authenticity. The user is not involved in the decision-making process at all, since the verification is automatically done by the browser. The visited host's certificate is accepted if it is signed by a CA that is present in the pre-defined list of trusted roots, for the right hostname, and neither expired nor revoked.

In recent years, a long streak of emerging breaches, inconsistencies, bad practices and mistakes at CAs, culminating at the total compromise of the leading Dutch CA *DigiNotar* [Hoogstraaten et al., 2012; von Eitzen, 2011, 2012], led to public recognition of the sorry state this established system is

in [Froehlich, 2012; Goodin, 2011a; Marlinspike, 2011b]. For selected publicly reported incidents, refer to [Baumgartner, 2012; Comodo, 2011; Goodin, 2011b; Hallam-Baker, 2013; Henning, 2012; Keizer, 2011; Menn, 2012; Tung, 2011; Walker-Morgan, 2012; Wisniewski, 2013; Zetter, 2012], to name just a few.

Many of the inherent systemic issues of this approach have been known for a long time [Clarke, 2001; Ellison and Schneier, 2000; Hayes, 1998; Moreau, 1998], but were only recently given the attention they deserve, due to the number of reported incidents. Furthermore, these incidents revealed that CAs might hesitate to report breaches in order to avoid harming their image. Since CAs are regular companies in a competitive environment, this is no surprise. For example, *VeriSign* kept breaches secret for about two years, until the U.S. Security and Exchange Commission (SEC) established new guidelines, forcing them to inform investors about incidents [Menn, 2012; Walker-Morgan, 2012]. Another CA, *Trustwave*, was caught deliberately selling subordinate CA certificates to companies for the purpose of spying onto their employee's communication to protect against intellectual property leakage [Baumgartner, 2012; Henning, 2012].

The reasons for ongoing concerns about the currently used PKIX approach are best described by illustrating the current state of the system in more detail. The Electronic Frontier Foundation (EFF) crawled the world-wide HTTP Secure (HTTPS) network in 2010 and provided the data on a platform called SSL observatory. Their data was gathered by scanning the entire allocated IPv4 space on port 443, the default HTTPS port. According to their results, there were 1.482 CAs with 1.167 distinct issuer strings, from 651 organisations. All of these CAs are trusted *by default* by software from Mozilla and Microsoft [Eckersley et al., 2010]. Included within the 1.482 CAs are both root and intermediate CAs. Intermediate CAs need not to be directly trusted by local software packages, since only the root is relevant.

Microsoft and Mozilla both maintain a public list of their trusted root CAs on-line, listing 353 [Albertson, 2013] and 158 [Mozilla, 2012] explicit certificates, respectively, with Microsoft having more intermediate CAs on their list. For an overview about the interrelations between root and subordinate CAs, we recommend the interactive explorer created by Amann et al. [2012][1], despite the smaller data basis.

All of the views are just snapshots of course. Like other companies, CAs are bought and sold from time to time, making maintaining such lists a tedious task and lowering the transparency even further. To put it with the words of Peter Gutmann: "*To take just one example of how confusing this can get, when someone tried to track down at least a few of the various keys belonging to the Comodo CA [...], they found that Comodo, a US company, had bought ScandTrust, out of Malmö in Sweden, who had in turn absorbed AddTrust from Stockholm, Sweden, and at the same time bought UserTrust from Salt Lake City, Utah who had certified Digi-Sign of Ireland (an SSL CA which is of interest primarily because it doesn't use SSL itself to secure access to its web-based accounts) later owned by the Dutch Getronics, and beyond that the spaghetti of sold, re-sold, and cross-linked CAs and keys became more or less impenetrable*" [Gutmann, 2013, pg. 633].

There is a large, complex and intransparent infrastructure hidden from users, which decides for them whether a host is trustworthy or not. Yes, users are warned if the website they want to visit cannot provide a certificate which is at least indirectly signed by one of the pre-installed roots. However, the user is not involved in deciding who is trustworthy, which, in the end, makes

[1] See http://notary.icsi.berkeley.edu/trust-tree/

the warnings less effective. Numerous studies evaluated the effectiveness of these Secure Sockets Layer (SSL) warnings, all of them concluding that they are mostly ignored [Bartsch et al., 2013; Herley, 2009; Sasse et al., 2012; Sotirakopoulos et al., 2011; Sunshine et al., 2009]. People are annoyed by them, do not know what to make out of the presented information, get used to them, and simply "click them away".

Moreover, there is *no mechanism* in place to avoid duplicates or to restrict the authority of an individual CA. Once trusted by a browser, a CA can issue valid certificates for any available domain in the world, even if another CA has already done so. The usual process for a service operator is to buy a certificate from one of these pre-trusted vendors to appear credible. Once obtained, the certificate is installed to make the web server deliver this certificate when clients visit the web site via HTTPS. However, this does not prevent a malicious or careless CA from issuing another certificate for *the same* host, completely out of the operator's control. The holder of such a duplicate certificate is able to impersonate an otherwise credible host without warning. Therefore, when colluding with a malicious or careless CA, it is possible to trick users into providing information they would otherwise give only to the real instance.

Combining the fact that all CAs around the world can issue certificates for arbitrary domains with the circumstance that selling certificates is the core business of a CA reveals a crucial systemic issue. Supposed that an adversary wants to maliciously impersonate a popular web page. This adversary can choose freely from the large pool of available CAs and try to trick it into issuing her a certificate. If one CA refuses, the adversary can simply move on to the next. Note that denying a certificate request is lost business for a CA. Thus, the probability that one CA might not look too closely is high, and security researches have already shown this multiple times [Nigg, 2008; Sotirov and Zusman, 2009]. For example, Sotirov and Zusman [2009] were able to obtain a valid certificate for Microsoft's `login.live.com` domain, simply by using the spoofed email address `sslcertificates@live.com` upon their enquiry. Matt Blaze summarized this problem back in 2000 with the famous quote, saying that "*a commercial CA will protect you from fraud by anyone whose money it refuses to take*" [Blaze, 2000]. The problem with multiple roots being predefined in web browser software is known since at least 1998 [Hayes, 1998] and, until very recently, no attempts to fix this came close to realization.

Interestingly enough, as Asghari et al. [2013] point out, this does not necessarily entail a *race to the bottom* scenario, where only cheapness counts and quality is of least importance. Differing certificate prices are justified with vague promises such as *increased security* or *warranties*, despite the fact that there is no known case where a CA actually covered losses caused by an impersonated host. What Asghari et al. noted is that enterprise customers seem to simply favor the most renowned Certificate Authorities, because they are considered *too big to fail*. Thus, host operators seem to be partially aware of the untrustworthy environment and opt for a choice that is not likely to be removed from the list of trusted roots, which would render their certificate purchase a bad investment. The whole CA agglomeration seems to boil down to a few key players on the market, who are profiting from this tendency towards major providers. According to Netcraft, Ltd. [2012], this is mainly Symantec, who bought leading CAs such as VeriSign, Thawte, GeoTrust and Equifax. The field of notable competitors in terms of market share is small, consisting only of GoDaddy and Comodo. Despite the conclusion that there is no race to the bottom scenario in terms of offered services, any attacker is looking to grab *lowest hanging fruit* in terms of security. Finding one trusted CA that does not follow best practices is enough to compromise all efforts by

those who behave as one would expect.

That certificate authorities do not always adhere to best practices is also shown by large-scale SSL studies, where certificates were examined regarding the used algorithms and their set properties. A notable subset of them shows severe flaws, be it in the use of outdated algorithms, such as Message Digest 5 (MD5), too small moduli lengths for RSA or even lots of duplicate key pairs, resulting from bad random number generators [Amann et al., 2013; Eckersley et al., 2010; Holz et al., 2011; Lenstra et al., 2012]. This is only partially to be attributed to CAs, since the key pair is generated at their clients. However, some CA's own root certificate keys are amongst the flawed ones, and probably kept in place since replacing a certificate at the root of a hierarchy for millions of other certificates is a non-trivial task.

While there is no doubt that the majority of CAs abide by their own rules and the policies defined by the software vendors, it simply does not matter as long as there is at least one CA in the very same pool that does not. The range of known incidents extends even further, namely to actively misbehaving CAs to gain further profits. *TrustWave*, a renowned CA, deliberately issued Man-in-the-Middle certificates to well-paying companies, enabling them to eavesdrop on their employees, as a means for protecting against intellectual property leakage [Baumgartner, 2012; Henning, 2012]. Given that CAs are sometimes government operated, it is thus a small leap to extensive spying on citizens, rumors of which exist since the last couple of years [Soghoian and Stamm, 2012].

The straw that broke the camel's back in a two-year series of negative events was the successful break-in to the formerly renowned Dutch CA *DigiNotar*. The break-in enabled adversaries to issue valid certificates for `mail.google.com` and `update.microsoft.com`, amongst others [Hoogstraaten et al., 2012; von Eitzen, 2011, 2012]. DigiNotar, which issued certificates for many Dutch government services at this time, was removed from the software vendors' list of trusted CAs, which immediately broke its whole business model. As a consequence, the Dutch government took over the CA and shut it down shortly thereafter [Binst, 2011]. Nevertheless, as migrating lots of services to a new CA takes time, the DigiNotar root certificate was moved to be a subordinate of an Entrust root CA[2] and remains active to this day, despite the incident being two years old and the company behind it not even existing any more.

The point is, however, that having TTPs, which are no longer to be trusted, puts the system into a broken state. To overcome existing flaws in this system, numerous add-ons, operating on top of the existing infrastructure, have been proposed. Most of these approaches address the most pressing problem of the currently adopted approach: the fact that CAs, located anywhere in the world, are technically able to vouch for any arbitrary domain. Notable extensions to the existing PKIX approach are discussed in Section 3.2.

1.2 Regulation Attempts

The current debate has also received attention from jurisdictions and the studies of law [Arnbak and Van Eijk, 2012; Asghari et al., 2013; Roosa and Schultze, 2013]. Aside from technical approaches to fix the situation, there are proposals to mitigate the situation by regulating Certificate Authorities

[2] As can be seen by searching for DigiNotar in the ICSI SSL Notary explorer: `http://notary.icsi.berkeley.edu/trust-tree/` (August 2013)

[European Commission, COM(2012) 238/2, 2012]. Currently, CAs are somewhat self-regulated, based on private arrangements between the involved institutions, for instance via the CA/Browser forum[3]. The current regulation proposal aims to make CAs *liable* for any damage that occurs because they are not adhering to "state of the art" security measures, and puts the burden on them to prove that they actually did. This might act as an incentive to invest in powerful logging mechanisms and stay up to date regarding key lengths and security processes. Furthermore, among other requirements, the regulation defines security breach notifications, which have to be sent "*within 24 hours, where feasible*" in case of breaches having "*severe impact on the trust service provided and on the personal data maintained therein*" [European Commission, COM(2012) 238/2, 2012].

However, it is unlikely that a single, failed CA can cover all recourse claims for, say, an unauthorized certificate for `mail.google.com`. Such a regulation might also hinder new players from entering the market, since they won't be able to afford insurance against such cases. Thus, it might further favor established providers [Arnbak and Van Eijk, 2012].

EU-level regulations can only regulate CAs within its jurisdiction. Every CA is currently able to vouch for any domain, regardless of any geographic or juridic bounds. Therefore, an EU-wide regulation is not considered sufficient, since it is a global problem. Even better would be a technical solution that does not allow this kind of abuse in the first place.

1.3 User-centered Security

STUNT is in line with ongoing efforts to actually include users into security-related decisions [Adams and Sasse, 1999; Furnell and Clarke, 2012]. Designing systems to have users decide in security-critical situations, without annoying them, is challenging, but we expect that this approach helps to raise awareness and thus increase understanding. Certainly, the decision-making process has to be supported with information understandable by novice users.

The conservative approach to embedding security mechanisms into an application is to hide the whole process from the user, called *implicit security* [Smetters and Grinter, 2002]. The concept behind implicit security is, to make systems usable by not interfering at all. Users are considered incapable or unwilling to make security-related decisions and should only be bothered when there is absolutely no other way. Currently, SSL is such a system, which hides all the encryption negotiation, integrity protection, and authenticity verification from the user. We also consider it useful not to have users manually configure all necessary details, such as selecting a cipher suite upon every SSL connection. Such an approach would lead to a severe overhead in workload without significant gain. In terms of authenticity, however, we think that users should well be included, to gain a fundamental understanding on what they are dealing with. The simple padlock, which is currently used in browsers to distinguish a "secure" page from an "insecure" one is a massive oversimplification of the complexity behind it. The padlock icon, which emerged from a set of web security user interface guidelines from the W3C [Roessler and Saldhana, 2010], is designed to not overwhelm users and provide the most simple way of indicating a secure connection, suitable for as much users as possible. The question is, however, how users are supposed to give a reasonable

[3] https://www.cabforum.org/

answer to a suddenly popping up SSL warning, if they were never confronted with the problem that a host might not be what it claims? No wonder that "*given a choice between dancing pigs and security, users will pick dancing pigs every time*" [Felten and McGraw, 1999] in terms of SSL warnings.

We believe that it should not be made that simple for users, by hiding all the dirty mechanics away from them, and present a green, shiny padlock that hints towards absolute security. This is especially the case when the user is the one being harmed when the padlock suddenly fails. Analogously, in a law context, people are well aware that not knowing about a certain regulation is no excuse for its perpetration. Here, people are well aware of possible consequences, albeit not in every detail. Therefore, people educate themselves before entering unknown legal terrain. For example, it might be a good idea to read up on building regulations, or include someone knowledgeable in the decision process, before starting to build a house.

In a network security context, people are typically not even aware that there is no absolute guarantee for authenticy. It "just works", with the browser and hidden infrastructure performing all the magic. Thus, there is no obvious incentive to gather information about what is actually going on behind the scenes. If it fails, and the current approach of dealing with the problem might lead to failures, how are users supposed to detect the problem? Network security is not just green and red, as often indicated by user interfaces. The dangers lie in not communicating this problem to the average user at all.

1.4 A Primer to STUNT

STUNT uses the simple mechanism of checking references to assess a host's trustworthiness. To do so, each STUNT-capable host shows a list of established relationships to reference hosts. These reference hosts are limited in their number to increase the value of each single trust relationship. Individiual references are not weighted and cannot be differentiated regarding its purposes. It is a mere *mutual* statement that both host operators assure the binding between host and host operator for each other. Trust relationships are means to authenticate a host, based on the set of references provided. The establishment of trust relationships requires the acknowledgement of both involved host operators. Therefore, a trust relationship can be understood as the expression of a host operator, stating that the trusting host is indeed operated by whom it claims to be. STUNT is therefore a method to authenticate hosts, in the sense that a host is indeed operated by the claiming party. It is comparable to the simple decision about whether or not to buy goods from a yet unknown, possible fake shop. The host's website is the equivalent to the store front, and the presented references are comparable to what others said about the shop in question, except that references in STUNT do not qualify as good or bad, but are simple assurances regarding the genuineness of a host. The mutuality shall ensure that these references are honest, since every outgoing trust relationship is incoming at the same time. Furthermore, referencing someone not trustworthy occupies a precious slot of available references. STUNT is designed to have all trust relationships verifiable by cryptographic means, to ensure that trust relationships presented to the user are not fake.

Checking references before engaging with an unknown entity is no absolute guarantee for fail-

safety. In contrast to existing approaches, however, the risk of falling for a faked host is clearly communicated. The user decides whether or not to trust a certain host, based on its references. More precisely, a user decides if the presented *cluster* of interrelated hosts makes sense. The cluster is expected to be a reflection of real world interactions between the host operator and its partners. Therefore, users, who are assumed to have at least a vague idea about the operator of the host they are about to visit, are considered able to assess the plausibility of the network. If the user already trusts a connected host, it is highlighted by the system, indicating that there is an established trust relationship between the host in question and another, already trusted one. If, however, the user ends up at a trust network where every host is entirely unknown, the system is bluntly telling the user that she is about to interact with an entity that is entirely unknown to her. If such a decision has to be made in the first place by the user, it should become apparent that there is the possible consequence of being wrong and trusting a fake host.

According to a 2006 Internet survey by Tickbox.net [2006] for the UK government, people use, on average, not more than six web sites on a regular basis. We think that this number might have increased by now, due to the advent of social media and user-generated content in general. Nevertheless, we consider it safe to assume that the subset of regularly visited web sites who really require time-consuming assessment is well below thirty, including on-line banking and secure workplace connections. One can surely use an encrypted channel to watch pictures of cats on the Internet as well, but authenticity is not a priority requirement there. In such cases, the evaluation can be omitted, since there is no sensitive data exchanged. The trust decision is remembered in a local database at the user's. Therefore, it has to be made only once, given that there are no changes on server-side. Assessing the trust relationships of the limited amount of websites which call for this measure is thus considered a feasible task.

STUNT is built upon the five design pillars explained briefly below:

User-central	Users are the ones who decide whether a host is trustworthy or not, and there exist individual trust bases per user or software component.
Trust Relationships	Trust relationships are an expression of confidence by an entity, that the recipient is indeed what is claimed. The authenticity of a host is assessed by its trust relationships. Users decide whether a certain cluster of trusting hosts is reasonable or not. Trust relationships from users to servers are unidirectional. Users trust hosts, not vice versa. Between servers on the other hand, only mutual trust relationships exist.
Expensive Creation	Trust relationships between servers are intentionally expensive to create, as incentive for investing only in long-lived relationships. Furthermore, the number of trust relationships is limited, so that the individual weight of a trust relationship is higher. Host operators are thus encouraged to choose their trustees carefully.
Cheap Verification	Users are able to verify the presented relationships on the fly.
Risk-based Revocation Support	The system uses a whitelisting approach, where no host is considered valid, except all of its trust relationships verify correctly

	and are current. An active revocation scheme is built-in, which requires continuous updates in predefined, risk-based intervals for a relationship to be kept alive.
The Backfiring Principle	As far as possible, all attempts to misbehave shall backfire at the initiator.

In STUNT, users assess the trustworthiness of a host by means of exploring the surrounding network of trust. Their decision is made within the browser's user interface and leads to individual trust bases, stored on the local client. Trust relationships are created using a simple protocol, similar to Secure Shell (SSH), which requires host operators to ensure each other's authenticity using a second channel. Trust relationships are created from exchanging a signed and thus sealed data structure. This data structure includes the certificates of the involved hosts, a guarantee for its freshness, and what's called a *proof of work*. The *proof of work* ensures the properties *expensive creation* and *cheap verification*. It is a cost function involving finding a certain partial collision for a secure, pre-image resistant hash function. The difficulty of this cost function increases with the amount of trust relationships of the *other* involved party. This asymmetry in effort is an incentive for host operators to actually engage in trust relationships with new, yet less trusted hosts. The mutuality of trust relationships embeds the backfiring principle, in a way that one's own reputation is harmed by careless trust decisions. Risk-based revocation support is realized by requiring updates to be sent in regular intervals for the trust relationship to stay alive. These intervals are risk periods, defined individually by both involved host operators during the establishment of a trust relationship.

Figure 1.1 shows an example of such a STUNT network of trust. The user, in the process of connecting to `tugraz.at`, is presented with `tugraz.at`'s trust network. This surrounding trust network shows all hosts, whose operators expressed their confidence that the host `tugraz.at` is indeed operated by the claimed Graz University of Technology, and vice versa. It is now up to the user to decide if `graz.at`, `universitaeten.at` and `online.tugraz.at` are plausible vouchers for the authenticity of `tugraz.at`. Moreover, if the user is unsure about any of the linked vouchers, the network can be arbitrarily expanded and browsed deeper, to assess the trustworthiness of shown trustees as well. A decision made by the user is then stored on the client, such that users will no longer be bothered in the future. In the background, however, the client will verify if the state of the trust network is still the same as it was during the assessment. If a critical deviation occurs, for example due to a revoked trust relationship, the user is warned and prompted to re-assess the trust network.

Compared to existing solutions, STUNT is certainly more work for both host operators and users. In contrast to PKIX, the responsibility of ensuring the authenticity of a certain host operator is shifted from CAs to other host operators. Users on the other hand have to decide for themselves if they trust in the authenticity of a certain network of trust configuration. This increase in effort at the user's might be the biggest barrier for STUNT, since its usefulness is highly dependent on whether or not users accept this way of assessing hosts. To mitigate this problem, STUNT is remembering the decisions and keeps the interruptions at a minimum. Nevertheless, it is more work for a user than in PKIX, which decides about authenticity without user interruption. Still, we think that it is essential for users to have a say in this decision and not have it taken away by an unknwon authority and intransparent processes. From a computational point of view, STUNT

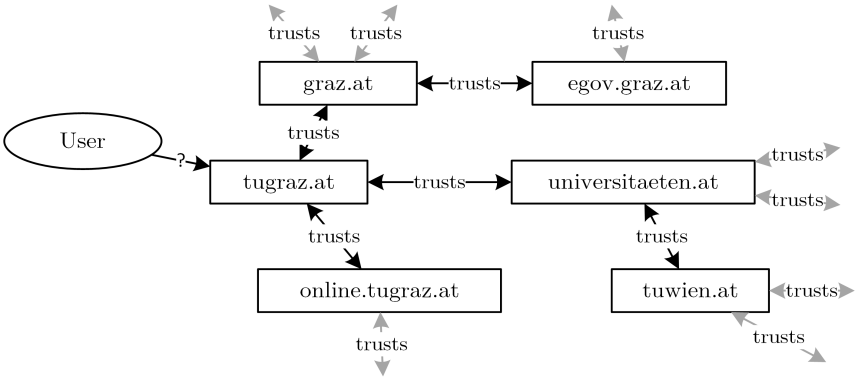


Figure 1.1: STUNT example network

is more work for host operators, since they are required to compute expensive proofs of work.

Deploying STUNT on the Internet would certainly be a major challenge, even though a backwards-compatible way is introduced. The backwards-compatible mode, which allows operators to use STUNT in conjunction with CAs still requires an additional key pair for host operators, of which the public key has to be included into the SSL certificate by means of an extension. Therefore, it is necessary to re-issue existing certificates and have CAs agree to the inclusion of this additional field. Moreover, it requires host operators to run an additional service on their hosts and to actively engage in trust relationships. Browsers or other clients have to include a module capable of verifying trust relationships and showing a network explorer component in an appealing way. Therefore, we are aware that it is considerable effort to introduce STUNT to a broad user spectrum.

While the concept is considered stable, making STUNT a ready for use requires future work regarding its parametrization. It is a prototype, and as such considered a solid base for further research. A new solution to the introduced problems is very hard to adopt, due to the vast amount of work that already went into establishing the current system and expected refusal to abandon the current way of dealing with the problem of host authenticity. Besides, there is a whole industry behind selling trusted certificates, which is not going to simply give their business model away to new approaches, which are no longer relying on them. To provide a way to theoretically introduce STUNT on top of the existing PKIX infrastructure, it may well be operated in a backwards-compatible way, using the established format of X.509 certificates and in addition to CA-based authenticity verification.

1.5 Outline

Before taking a more detailed look at STUNT, preliminary topics are covered in brief. Afterwards, we discuss existing approaches towards ensuring host authenticity. Background knowledge about these solutions is needed to understand their individual advantages and disadvantages. Moreover, parts of their concepts are inherited in STUNT, whereas other aspects were intentionally designed to be different. In chapter 4, we discuss STUNT in more detail, covering technical aspects as well as internal workings. Subsequently, the STUNT prototype implementation is introduced, with an overview about implementation details and an initial user interface design idea being shown. Before

drawing final conclusions, the system's advantages and disadvantages are discussed in Chapter 6, and the achieved properties are compared to the goals defined in Section 4.1.

Chapter 2

Preliminaries

"The trust of the innocent is the liar's most useful tool."
– Stephen King

This chapter briefly introduces concepts and notations required to understand the following discussion about STUNT and related systems.

2.1 Secure Communication in Large-Scale Networks

Enabling secure communication over large-scale networks, whose individual components are controlled by different parties, is a challenging task. Secure communication incorporates the three key pillars defined by the *CIA triad*: *confidentiality*, *integrity* and *availability* [ISO/IEC 27000, 2009].

Confidentiality, protecting message data in a way that avoids eavesdropping by other parties, can be achieved by encryption. Encryption, however, introduces the problem of key distribution. Given two parties who want to communicate over a network that is considered insecure, how are they supposed to exchange keys to use them later for en- and decryption? Asymmetric or public key cryptography eased this situation by introducing key pairs, where a publicly available part is used to encrypt the message. Alongside this public key, each participant is in possession of a private key that must be kept secret. A message encrypted with a person's public key can be decrypted with the corresponding private pendant. Public key cryptography schemes such as RSA [Rivest et al., 1978], ElGamal [El Gamal, 1985] or the Elliptic-Curve Integrated Encryption Scheme (ECIES) [Brown, 2009] can be used to achieve confidentiality.

Integrity, the second aspect of the CIA triad, concerns both *message integrity* and *origin integrity*. Message integrity refers to the message not being altered by an intermediate instance. Origin integrity, on the other hand, deals with the assurance that both communicating parties are indeed who they claim to be. Both these attributes are usually achieved by either digital signatures [Brown, 2009; El Gamal, 1985; Rivest et al., 1978] in asymmetric cryptography scenarios, or by Message Authentication Codes (MACs) [ISO/IEC 9797, 2011], if symmetric cryptography is used.

Availability refers to whether the involved systems are available to the participants. A system that is turned off, or disconnected from the outside world might be hard to break into, but its usefulness is limited as well.

There is ongoing criticism on the *CIA triad*'s simplicity, and extended models such as the *Parkerian Hexad* [Parker, 1998] exist. However, we consider the CIA approach sufficient for a brief introduction on the relevant attributes of secure, digital communication in this context.

Cryptographic assurance in terms of confidentiality and integrity is indispensable for secure communication. However, at some point humans are involved, which adds the additional problem of binding a certain secretly kept key to a specific person. Each key has a particular purpose and crypto systems are designed in a way that renders guessing keys from observing ciphertexts is computationally infeasible. In the end, however, it is just a long, random number with no guarantee that the right person is in its possession. This is particularly problematic on first encounters in a large network scenario, where participants typically do not personally meet or maybe do not even know the other endpoint. A newly established communication channel might thus ensure confidentiality as well as message integrity, but lead to a different, non-intended person.

Several approaches exist to mitigate the uncertainty of whether or not a key belongs indeed to a certain person. Existing concepts are introduced in Chapter 3.

2.2 Trust

Before discussing existing approaches to ensuring host authenticity, there is need to elaborate on the term *trust*. There is no generally accepted definition of trust and there are many facets to that topic [Das and Teng, 2004; Grandison and Sloman, 2000; McKnight and Chervany, 1996, 2001], such that it makes sense to outline the diversity of the term trust. Within Section 4.2, we will give a definition of what *trust* means in the context of STUNT.

2.2.1 Trustable Entities

Jøsang defined that there is a distinction to be made on whether the trusted party is *passionate* (human) or *rational* (non-human). His main argument satisfying this distinction is that a human entity might be intendedly abusing one's trust or be subject to threat or persuasion. Therefore, according to Jøsang, trusting a human *"is the belief that it will behave without malicious intent"*. Interpersonal, *passionate* trust is a complex topic on its own [Bamberger, 2010], but considered out of scope in this context. Non-human entities are not able to pursue malicious goals by themselves, because they are designed for a specific purpose. Trust in such a rational system is therefore denoted as *"the belief that it will resist malicious manipulation by a passionate entity"* [Jøsang, 1996]. This definition does not take malicious designers of rational entities into account, who create components whose entire purpose is to follow malicious intents. These components might resist malicious manipulation by outsiders, but are malicious by itself.

X.509 defines *trust* as follows: *"Generally, an entity can be said to 'trust' a second entity when it (the first entity) assumes that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function."* [ITU-T, 2009]. Note that this definition does not imply whether the system or other party behaves in a subjectively positive or negative way. Furthermore, this definition implies a thin line between trustworthy and reliable, since reliability is also defined by expected behaviour, although with positive connotation. From these two sample definitions, we see that trust differs strongly, depending on the expectation someone has towards the entity to be trusted, and its intended purpose.

Somebody being trustworthy, on the other hand, means that *"one is able and willing to act in the*

other person's best interests" [McKnight and Chervany, 1996]. Therefore, it does encompass the positive attitude towards the trustee.

In a distributed environment, where personal visits are hardly possible, one has to be sure to actually talk to the intended system. Therefore, trust in that context can be twofold: a) whether one trusts in that the other host is indeed operated by whom it claims to be, and b) whether to trust that particular host operator per se. The X.509 definition of a trusted system blurs this distinction by stating that the system behaves as expected. One could actually argue that *expected behaviour* is a subjective and hardly verifiable attribute. While it is usually apparent if a system does the opposite or less than what is expected, one is typically unable to detect whether it is doing more than that. For example in phishing scenarios, where websites are exactly copied in order to behave as if they were real, just to additionally grab users' credentials and send them to some place else.

2.2.2 Trust Directionality and Qualification

Reasonably, it is common ground in the literature that trust relationships are unidirectional [Abdul-Rahman and Hailes, 1997; Denning, 1993; Golbeck, 2006; Jøsang, 2013]. Furthermore, many proposed trust management approaches additionally qualify such relationships. This is done by either quantifying the amount of trustworthiness accredited to some entity [Abdul-Rahman and Hailes, 1997; Zimmermann, 2000] and thus addressing the uncertainty of such a relationship, or by classifying the purpose of a certain trust relationship [Blaze et al., 1999, 1996; Chu et al., 1997; Clarke et al., 2001; Yahalom et al., 1993]. The latter includes categorization, such as whether an entity is trusted for the purpose of car servicing, stomach surgery or financial consultancy.

2.3 Proof of Work

A proof of work is the output of a mechanism to prove to someone else that a substantial amount of work was performed when fulfilling a certain task. Proofs of work are designed to be hard to provide, but cheap to verify at the same time. Proofs of work are used as spam protection mechanism [Back, 2002], as *client puzzles* to prevent Denial-of-Service attacks [Juels and Brainard, 1999] and are currently heavily computed during *BitCoin* mining [Nakamoto, 2008].

The idea is to take a preimage-resistant hash function, such as Secure Hash Algorithm (SHA)-256 [Dang et al., 2012], which is computationally infeasible to invert. Instead of finding a complete preimage, the problem is reduced to finding a certain input, which produces an output hash *smaller* than a defined number. The problem's difficulty can be adjusted by the required size of the output hash. Typically, the upper output constraint is a value evenly divisible by two, such that *smaller* can be expressed by the number of leading zeroes of the output hash. With SHA-256 being used, the problem's complexity scales with 2^n, with n being the number of required leading zeroes [Dang et al., 2012].

2.4 Hash Chaining

Hash chaining is a method to prove the integrity for a list of entries in an order-preserving way. To achieve that, successive computation of a hash over the list of entries is performed in the following way:

$$hashchain = H(... \mid\mid H(\text{entry2} \mid\mid H(\text{entry1})))$$

At first, a hash of the first entry is computed. Each following entry is concatenated to the output of the previous hashing operation. This concatenated byte string is used as input for another hash computation. The output of this hash computation is then concatenated again to the next entry and hashed, and so on. This sequence of computations is repeated for all entries. The resulting output is a fingerprint encompassing all entries of the list, including their order. Therefore, assuming a secure, preimage-resistant hash function, this fingerprint is only reproducable using the same entries in the same order again. The mechanism is the same as in secure boot scenarios, where Platform Configuration Registers (PCRs) are updated in the same manner [Trusted Computing Group, 2011, Chapter 4.4]. In trusted booting, hash chains are used to determine if two consecutive system boot-ups were executing the very same software in the same order. Thus, potentially malicious deviations in the host's software configuration are detected.

Chapter 3

Background and Related Work

"Ginny!" said Mr. Weasley, flabbergasted. "Haven't I taught you anything? What have I always told you? Never trust anything that can think for itself if you can't see where it keeps its brain?"
– J.K. Rowling, Harry Potter and the Chamber of Secrets

STUNT is a new approach to address the problem of host authenticity in large-scale networks, where consumer and provider typically don't know each other. Therefore, it involves establishing a binding between a host and its operator. The proposed system is, to the best of my knowledge, a new approach towards the host authenticity problem. There are already several approaches for this problem, all of which have their own advantages and disadvantages. Since STUNT is inspired by these systems and shares parts of their concepts, relevant systems are discussed and compared in this chapter.

Research on this topic was very active up to about the turn of the millennium, propelled by sceptics of the adopted PKIX solution like Matt Blaze [1999; 1996], Stefan Brands [2000], Roger Clarke [2001] or Bruce Schneier [2000]. Additionally, the World Wide Web Consortium (W3C) was working on a rule-based alternative called Rule-controlled Environment for Evaluation of Rules, and Everything Else (REFEREE) [Chu et al., 1997]. Unfortunately, most of the early alternatives were neither developed any further nor widely used and are thus no longer of practical relevance. Pretty Good Privacy (PGP) [Zimmermann, 2000] on the other hand, which was created at about the same time, is still used for private e-mail communication. Anyhow, for host authenticity, the almost solely used solution is Public Key Infrastructure X.509 (PKIX) [IETF PKIX Working Group, 2013], a family of standards and Requests for Comments (RFCs) for public-key infrastructures. STUNT is designed to be operated in both a PKIX-compatible and a standalone mode. Both the popularity and compatibility of PKIX justify a deeper delve into its workings, followed by PGP, Simple Public Key Infrastructure (SPKI), Domain Name System Security Extensions (DNSSEC) and a more general look systems establishing trust by reputation.

3.1 PKIX

PKIX [IETF PKIX Working Group, 2013] is a historically grown, vast suite of standards and RFCs, defining various aspects of establishing a public-key infrastructure in an Internet context. For the sake of conciseness, only the directly relevant parts are to be described throughout this section. More detailed information can be found either in the ITU-T X.509 standard itself [2009], Adams and Lloyd [2002] and Karamanian et al. [2011], RFC 2459 [1999] for the certificate structure, as well as the RFCs 5280 [2008] and 2560 [1999] for detailed revocation-related information.

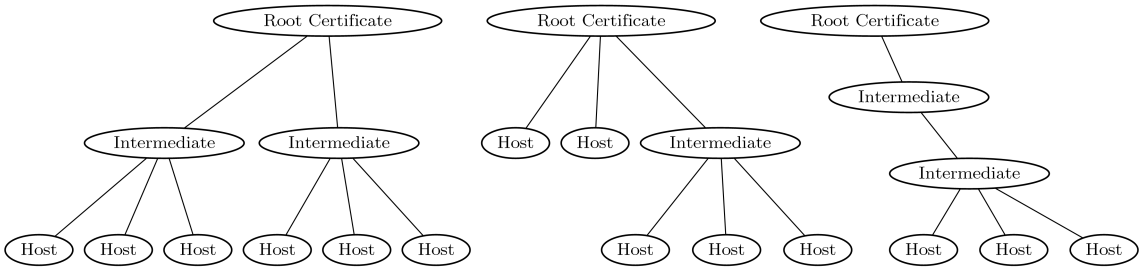


Figure 3.1: Certificate hierarchy examples

3.1.1 Introduction

PKIX introduces TTPs, or so-called Certificate Authorities, which are to be trusted by both the service provider and the end user. In this setup, every service provider owns a self-created public/private key pair. To be accredited by a Certificate Authority, a service provider signs its public key using its own private key and sends it to the CA, including some identification, for example a passport, and receives a *certificate* in return. A certificate is basically the public key of the service provider, signed by the CA. More precisely, a certificate contains meta-information such as a validity period, the allowed key usage and possible constraints, for example restrictions on what the certified key may be used for. The certificate structure will be explained in more detail in section 3.1.2.

The signature by the CA is a seal, stating that the service provider in possession of the corresponding private key is indeed who is being claimed. Furthermore, the signature prevents later altering of the certificate information.

Central to PKIX is the hierarchical certificate system, as defined in X.509 [ITU-T, 2009]. At the top of each hierarchy, a CA's certificate resides. A root CA's certificate, a so-called *root certificate*, is self-signed. Since there is no superordinate instance, the CA signs its own public key to prevent later modifications. Further down the tree-like hierarchy are either direct leaves, or subordinate CAs. Subordinate CAs can issue further certificates, which in turn reside below the subordinate CA's level in the hierarchy. Whether a node is a subordinate CA or a host is defined by the values of a triplet of certificate fields, the *key usage*, the *basic constraints* and the *extended key usage*. Figure 3.1 shows possible sample hierarchies. These hierarchies exist in parallel, with a different rootCA residing at the top of each tree.

This hierarchy-based approach implies the assumption that anyone who trusts the root automatically trusts all the other parties whom the root vouches for. Therefore, the overhead at the end user of manually selecting trustworthy hosts is greatly reduced. In fact, even the choice of trustworthy roots is taken away from end users. Nowadays, typically each vendor of software using this system, for example browsers, operating systems or runtime environments, includes a list of trusted roots. Therefore, software vendors ultimately decide who is trusted *out of the box* and who is not.

A Certificate Authority is usually a company, either privately operated or driven by a government. The business model of such CAs is to charge for issuing certificates. Certificates are are delimited in their lifetime by the validity period, thus forcing clients to periodically buy renewals. CAs have to ensure that the recipients of issued certificates are indeed who they claim to be, and also have

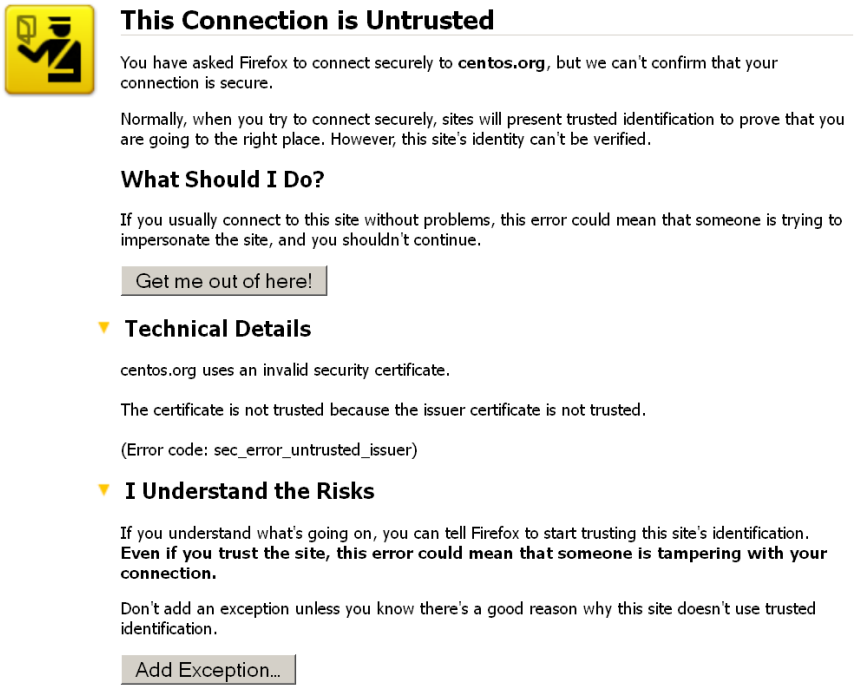


Figure 3.2: A sample SSL warning dialog in Firefox

to ensure adequate protection of their own key material to avoid abuses caused by leakage. For a CA, it is crucial to remain *trusted*, that is, within the set of predefined CAs that are shipped with browsers and other software packages. Being trusted out of the box is essentially a CA's only relevant argument for customers to acquire certificates. If the CA were not in the list of trusted roots, anyone could sign a certificate with the same effect, namely a warning at the end user's, such as the one depicted in Figure 3.2. Once a CA is trusted, it can issue valid certificates for every arbitrary existing domain.

Prices for acquiring certificates range from $25 to $1000, depending on the type of ownership verification performed [Asghari et al., 2013]. A Domain Verification (DV) certificate, the least expensive type of certificate, is only verified by sending a verification email to the email address specified in the WHOIS entry of the corresponding website. Extended Verification (EV) certificates, which involve detailed checking of the owner by the CA, are at the upper end of the given price range. From a user experience point of view, the only difference between less expensive DV certificates and higher-quality EV certificates, is simply that the name of the associated organization is shown in the URL bar of the browser. The difference is illustrated in Figure 3.3, note the part left of the URL in Figure 3.3b.

(a) DV certificate sample (b) EV certificate sample

Figure 3.3: The difference between DV and EV certificates as shown by Mozilla Firefox

PKIX is not limited to service providers, since end users can obtain certificates as well. Examples for use cases directly involving end users would be client authentication or secure emailing (S/MIME). The further description focuses on the relationships between service providers and end users, since it is the relevant aspect for this thesis.

3.1.2 Certificates

The certificates mentioned before are X.509 version 3 certificates. These structures contain at least the fields shown in Table 3.1. Certificates by itself are complex structures, with room for ambiguities and mistakes in their validation [Gutmann, 2002]. Especially Distinguished Names (DNs) turned out to be insufficient for unique identification [Kaminsky et al., 2010]. X.509 was originally intended as authentication layer based on a networked directory service, X.500. Due to its complexity, X.500 was never fully implemented. However, the Lightweight Directory Access Protocol (LDAP), a subset of X.500, is in wide use. X.509 was reused as an approach to host authenticity, but several adaptations had to be made to suit the resulting new needs. For this purpose, extensions were introduced with version 3 of the X.509 certificates.

Table 3.1: X.509v3 certificate fields [Housley, R. and Ford, W. and Polk, W. and Solo, D., 1999]

Field	Description
Serial Number	A unique identifier for the certificate, typically a 128-bit random number.
Version	Specifies the version of the structure. Currently, the values v1, v2 and v3 are defined.
Signature	Contains meta-information about the algorithm that was used when signing the certificate. Possible algorithm families, such as for example RSA in conjunction with SHA-256, are pre-defined by so-called object identifiers (OIDs).
Issuer Digital Signature	The actual signature, as created by the certifier
Issuer	(Uniquely) defines the issuer using a Distinguished Name (DN)
Subject	(Uniquely) defines the certificate owner using a Distinguished Name
Validity Period	Defines a time span during which the certificate is valid. It is defined using two timestamps, specifying the time before and after which the certificate is no longer valid.
Subject Public Key Information	Contains the public key of the owner, alongside an OID for the used algorithm
Extensions	Optional extensions, since v3.

Each extension is a triplet containing a *critical* flag, an Object Identifier (OID) as well as its value. According to RFC 2459 [1999], critical extensions must be recognized by compliant services. If a certificate contains an unrecognized but critical extension, it has to reject it. On the other hand, non-critical extensions may optionally be taken into account upon verification. The crucial certificate fields *key usage*, *extended key usage* and *basic constraints*, which control how a certified key may be used, are only critical extensions and not part of the original certificate structure. A brief explanation of these three extensions is given below, since they are important during verification.

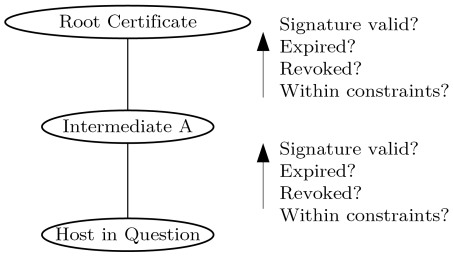


Figure 3.4: X.509 certificate chain verification [cf. RFC 5280 2008, sec. 6]

As mentioned before, the fields *key usage*, *extended key usage* and *basic constraints* define the intended purpose for an issued certificate. They enable the distinction between a CA and a leaf certificate. Furthermore, they allow restricting a certificate to a certain use case, for example to be used only for digital signatures, encryption or certificate signing. Extended key usage, in addition to key usage, allows issuers to place even finer grained constraints on the represented key. The basic constraints field explicitly defines whether the certificate is a CA certificate. Moreover, a maximum depth of the hierarchy tree is specified.

3.1.3 Verification

Verification is a complex topic on its own, with many opportunities to err [Georgiev et al., 2012]. Verification is discussed on a brief level, leaving out many peculiarities and possible side-effects.

In PKIX, a certificate is verified by walking up the certification path. When visiting a certain host, for example using the SSL protocol, this host will present its certificate. Since the visited host is most likely not a CA, its certificate should be signed by either a subordinate CA or a root CA. In case of a subordinate CA, this subordinate itself has to be signed by a higher-level instance, and so on. Figure 3.4 shows a sample verification path, alongside the individual checks that need to be performed on each hierarchy level.

Given a set of certificates that form such a path, signature verification can be done by using the public key of the direct superordinate certificate for verifying the signature on the currently processed certificate. Furthermore, the issuer name of the currently processed certificate has to match the subject name of the superordinate one. As a second step, the validity period is inspected to ensure the certificate is still valid. If the first two steps were successful, a revocation check is performed. Certificates might be revoked, for example due to abuse or private key compromise. Revocation is discussed in more detail in Section 3.1.4. As a last step, any constraints defined in critical extension fields have to be processed. The current path length has to be matched against the maximum allowed path length. Furthermore, the key usage field needs to be inspected to check whether an intermediate CA is indeed allowed to sign other certificates.

These four steps are repeated for each certificate in the chain. Once the verification process reaches the top, the root certificate is matched against the local, pre-configured set of trusted root certificates. If the presented root is such a trusted root and the whole path verified correctly, the certificate of the host in question is considered valid.

However, as the system grew it became increasingly slow in adapting to newly introduced changes,

leading to a multitude of problems with this seemingly straightforward verification procedure [Gutmann, 2013, pg. 652ff]. One potential cause for problems is that, in practice, the hierarchy is often no longer a tree. Cross certification, in all its variants, allows CAs to express their trust in each other, leading to certification paths that might contain loops or are otherwise ambiguous [Boeyen, S. and Moses, T., 2003; Gutmann, 2002].

3.1.4 Revocation

Revocation means the withdrawal of a certificate, even if it might otherwise still be valid in terms of its validity period and verification path. Reasons for revocation can be private key compromisation or a change of the domain name, for instance.

In PKIX, revocation follows a blacklisting approach. Blacklisting implies that, by default, every certificate is considered non-revoked until present on a Certificate Revocation List (CRL). Such Certificate Revocation Lists are typically issued by the certificate authorities themselves and are signed to avoid unauthorized modification. A specific key usage flag in the certificate specifies if a certificate holder is allowed to sign a CRL.

A CRL is merely a list of the serial numbers of revoked certificates. It is not intended to be on-line, and can thus be cached. For reasonable caching, each CRL contains a field specifying a date for its next update, after which requesters shall update their cached version. Furthermore, there are variants of CRLs, like delta CRLs, containing only the differences to another reference list, or indirect CRLs, which incorporate information of other CAs as well [Cooper et al., 2008]. Typically, the location of such a CRL is encoded within the certificates, as part of the CA policy certificate extension.

Revocation might be an urgent matter, and the concept of cached CRLs is thus being widely replaced by the Online Certificate Status Protocol (OCSP). OCSP allows questioning the revocation state of a certificate directly upon verification. Certificates from CAs supporting OCSP typically contain another policy extension field containing the location of the CAs OCSP server. This server can be queried with the certificate's serial number and issuer name, to get information about its revocation state.

Both OCSP requests and responses shall, but must not, be signed to prevent tampering with its contents. An OCSP response returns a status field, with the possible values *GOOD*, *BAD* or *UNKNOWN*. The only definitive answer, however, is *BAD*, which states that the certificate has indeed been revoked. *GOOD* might mean that the certificate is non-revoked, but might mean as well that the certificate was issued somewhere else, or "*that the time at which the response was produced is within the certificate's validity interval*" [Myers et al., 1999]. Therefore, it might as well contain no usable information at all. Furthermore, it is up to the implementation how to react upon *UNKNOWN* responses. OCSP responses do not guarantee the freshness of the response, since it is unknown where the server gets the state information from. It may well be that the OCSP server simply queries a CRL in the background, thus providing no better information than the CRL itself.

3.2 PKIX Additions

The approaches presented here are focused on backwards compatibility, to keep the maintenance overhead within reasonable bounds when transitioning from X.509 to something different. Therefore, they are designed as add-ons to the existing infrastructure, seeking to improve the current situation by providing more information at verification-time.

3.2.1 Public Key Pinning

Public Key Pinning (PKP) is an attempt by Google to enhance operator control about which CAs are authorized to vouch for their authenticity. To achieve this, an additional HTTP response header field `Public-Key-Pins` is introduced, where host operators can include a list of base64-encoded hashes of public keys from root certificates, which denote valid roots of trust for their domain. PKP is currently in review by the Internet Engineering Task Force (IETF). The summary given here is based on the currently most recent draft, revision six, valid until December 21st, 2013 [Evans et al., 2013].

A PKP-capable HTTP response header might look as follows:

```
HTTP/1.1 200 OK
Date: Sat, 29 Jun 2013 10:32:45 GMT
Server: Oracle-Application-Server-10g
Content-length: 5030
Content-Type:
text/html; charset=UTF-8
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Public-Key-Pins: max-age=3000;
                 pin-sha1="ZGEzOWE[..]Q4MDcwOQ==";
                 pin-sha256="ZTNiMGM0ND[..]k5MWI3ODUyYjg1NQ=="
```

This sample response header states that only the two root certificates with either a base64-encoded SHA-1 hash of `ZGEzOWE[..]Q4MDcwOQ==` or a base64-encoded SHA-256 of `ZTNiMGM0ND[..]k5MWI3ODUyYjg1NQ==` (both hash values shortened for clarity) are valid. The `max-age` directive is a time-to-live value for the information to remember (pin). On each visit, the browser remembers the current timestamp plus `max-age` in seconds, after which this header information expires and is no longer used as reference. Subsequent visits with equal PKP header information will update this value, further extending its relevance. Note that in HTTPS traffic, SSL protects both the header information and the content in terms of confidentiality and integrity, thus avoiding easy manipulation of the header information.

Therefore, Public Key Pinning allows host operators, who know the CAs they contracted with, to disseminate this information to visitors. Another CA, which, fraudulently or accidentally, issues a certificate for this host, will have a different hash of the root certificate's public key. Thus, a subsequent attempt to impersonate the host will be recognized by clients who visited the real host

beforehand, since the pinned information is different from the presented data. It does not, however, solve the problem on a first visit, since an impersonated host may as well send faked PKP headers.

Public Key Pinning intentionally uses public keys and not the whole certificate as reference, to allow new certificates to be generated from a reused key. To summarize, PKP enhances the control of service operators, since they are able to spread the information about what CAs to accept within their HTTPS responses. Users, having the intended keys pinned from previous visits, will be able to detect a certificate signed by a CA that is not authorized to do so by the host's operator on subsequent visits.

3.2.2 TACK

Trust Assertions for Certificate Keys (TACK) are currently under review by the IETF as well. The contents of this section are based on the current draft version two, which expires in July 2013 [Marlinspike and Perrin, 2013].

TACK is conceptionally very closely related to Public Key Pinning, but uses a second key pair, the TACK key, in addition to the SSL key pair. The TACK private key is used to sign the host's public key as used in the certificate. If requested by TACK-aware clients, the TACK public key as well as the signature are sent by means of a SSL/TLS extension. With the TACK public key, the SSL/TLS public key is verified. If the verification succeeded a defined number of times, the given information is remembered (pinned) until a given time frame expires. It has to be successful for more than one time, to avoid pinning faked information, which might be presented on the first, but not on subsequent visits. As with Public Key Pinning, the verification result will be compared on subsequent visits and the connection will be aborted or a warning will be issued if the results mismatch, for example due to an adversary performing a Man-in-the-Middle attack.

Contrary to Public Key Pinning, TACK is intended as standalone authenticity method, with just optional verification of any present X.509 certificate chain. TACK pins shall be easily shareable, thus allowing clients to verify their result by others posted on the Internet. The main difference between Public Key Pinning and TACK is therefore that TACK uses another key to bind to the SSL/TLS key, whereas PKP uses a binding to a public key somewhere up in the certificate chain.

However, TACK itself does not offer any reliable method to verify whether the originator is indeed who she claims to be. While it makes it exceptionally hard for attackers to perform a Man-in-the-Middle attack on a client who has visited the original host before within the timeframe, it does not provide means to verify the destination on first visits. This identification might be achieved with several, defined instances, who report the same pins upon requests, which is then very similar to Perspectives, being introduced in the following subsection.

3.2.3 Perspectives

Perspectives is an interesting approach introduced by Wendlandt et al. [2008], which got picked up and sharpened from a prototype state to a production-ready implementation by Marlinspike [2011a], called Convergence. Both approaches share the same concept, only being different in implementation-level details. Perspectives uses what Wendlandt et al. called *notaries* as a source

for authenticity information. Notaries are simple servers, which record the certificate fingerprints of other sites found on the web.

Contrary to the existing PKI hierarchy, notaries are not trusted by themselves. Instead, a quorum-based approach is pursued. Clients who visit a website using HTTPS will forward the host name and received fingerprint of the presented SSL certificate to a number of predefined notaries. These notaries will compare the received tuple with the corresponding record in their database, and tell the client whether the record matches or not.

Depending on the client's configuration, a specific quorum of positive notary responses is required for the host to be considered authentic. The quorum-based approach eliminates the possibility that a single entity can trick a client into accepting a host as valid. Clients who are subjected to a Man-in-the-Middle attack are assumed to receive negative notary responses, since the presented fingerprint will not match. Of course, this system only works as good as a client's notary choices. Recommended would be to choose at least three notary servers, which are preferably located across the world and known to be independent of each other.

The system comes with two interesting properties. The first one being that clients are not bound to the fixed CA which issued a host's certificate. Trustworthiness is assessed using the notary responses, and notaries can simply be changed by adjusting the client-side configuration. Therefore, "untrusting" such a notary is a simple task. The second property is, assuming that notaries are chosen well and placed across the world, that it gets very hard for attackers to fake a host. Notaries can of course be tricked in the same way into storing wrong fingerprints as clients. However, the quorum-based approach would require the adversary to have high control over the network. Even governments, who may operate a CA to spy on their citizens using valid, duplicate certificates for prominent websites, should be unable to influence the routes of relevant notaries located in very different places.

Both Perspectives and Convergence offer a Firefox plugin[1]. Unfortunately, notary servers are quite scarce at the time of writing, which harms the project's usefulness and hinders its wider use.

A similar approach to Perspectives is the P-Grid Public Key Infrastructure (PKI). It transfers the same, statistical concept to peer-to-peer networks, where an agent is considered trustworthy if a certain quorum of other agents acknowledge its public key [Datta et al., 2003].

3.2.4 CertLock

CertLock is a browser extension proposed by Soghoian and Stamm [2012]. It collects and stores metadata of the root certificate at the top of the presented certificate chain when the client connects to a website using HTTPS. Collected are values such as the country of the issuing CA, the CA's name, as well as name and country of the website itself. The collected data is stored alongside the host name, and the information received on subsequent visits is compared to a previously stored record, if present. If a property changed, a warning dialog is shown, raising the user's awareness that something unexpected is going on.

The rationale is that many governments operate top-level or subordinate CAs. Therefore, it is

[1] See https://addons.mozilla.org/en-us/firefox/addon/perspectives/ and http://convergence.io

possible to issue valid certificates for arbitrary domain names to spy on citizens. If a country's Internet traffic is routed through some Man-in-the-Middle infrastructure, presenting such a crafted certificate, users might not notice it as long as the root CA is in the global trust store. However, if one happens to travel a lot and visits the same website from different countries, then the browser remembers the corresponding certificate chain and its attributes. Thus, if one is at another destination and the chain evaluates to another top level root, users receive additional warnings. Typical browser implementations won't warn about this out of the box, because no state is maintained across subsequent visits and the root is still a globally trusted one.

3.2.5 Certificate Transparency

Yet another current IETF draft by Laurie et al. [2013] proposes *certificate transparency*. The contents of this section are based on revision 12 of the draft, which expires in October 2013.

Certificate transparency introduces the idea of keeping global logs containing identifiers of issued certificates. Clients are expected to check the log for any presented certificate they encounter and reject them if they do not appear therein. Responses from the log are lists of hashes, signed by the log, which prove that a certificate is indeed recorded. Thus, CAs are forced to publish issued certificates to the log. The log itself is a service built around an append-only Merkle hash-tree [Merkle, 1988], with revocation being delegated to other, existing mechanisms, like CRLs.

Certificate transparency defines two roles: *monitors* and *auditors*, where monitors continuously watch over the log to ensure consistency, and auditors query the log for its contents. An auditor component can be built-in into a web browser, for example. It possesses partial information from the log, namely the currently questioned certificate, and queries the log to ensure the certificate was published there.

CAs need to report newly issued certificates to the constantly monitored log, certificate transparency avoids duplicate certificates for the same domain. Therefore, conventional SSL Man-in-the-Middle attacks are counteracted. Logs can be operated by anyone, and need not be trusted by clients. The idea is to have a pre-defined set of trusted logs, much like predefined roots of trust.

A somewhat similar attempt driven by the EFF, called *sovereign keys*, also incorporates a central data read- and append-only data structure. This distributed data structure, spanning ten to twenty servers and being mirrored even more often, allows only one entry per domain. Legitimate host operators are supposed to create a key pair (the *sovereign key*), and append its public part to that centralized structure. Since there is only one entry per domain allowed, duplicates are avoided, and attackers able to obtain a valid certificate for a domain from a misbehaving CA cannot insert the required record into that structure. Clients can query this structure for its corresponding recorded entry when visiting a host, and compare the returned data with the certificate returned from the host.

The crucial difference to certificate transparency is that sovereign keys has the stated goal to completely remove SSL warnings due to their ineffectiveness (cf. Section 1.1). In sovereign keys, host operators can specify an alternative route to the target host, which is automatically used if the sovereign key verification fails at the client. Such an alternative route might use a Virtual Private

Network (VPN) connection or establish the connection via a hidden The Onion Router (TOR)[2] service, whose address is the hash of the sovereign key, ending with `.onion`. Therefore, the browser automatically routes around a possible ongoing Man-in-the-Middle attack without further user interaction [Eckersley, 2012].

3.3 PGP

Pretty Good Privacy (PGP) by Zimmermann [2000] is a well-known counterpart to the hierarchy-based approach of PKIX. It never received the same amount of attention as PKIX did, but it is still widely used for private messaging. PGP is based on a user-centric, decentralized model, a concept which was loosely adopted by STUNT.

Zimmermann created PGP in 1991 to promote awareness of privacy issues in the digital age. After the on-line publication of the PGP software, Zimmermann was subjected to a criminal investigation, since the US copyright export law classified crypto systems using key lengths longer than 40 bits as *munition* and restricted them from leaving the country. Zimmermann's response was to publish the entire PGP source as a hard-cover book [Zimmermann, 1995], allowing everyone to recover the software from scanning the book's pages. Despite the publicity boost arising from the court fights, PGP's popularity might have been hindered by usability problems, as pinpointed by Whitten and Tygar [1999]. The brief PGP explanation in this section is based on [Zimmermann, 2000].

3.3.1 Certificates

Certificates in PGP serve the same purpose as those in X.509, to bind a public key to an identity, but are constructed in a different way. PGP certificates, usually called just PGP keys, contain, without being limited to, the fields shown in Table 3.2. Contrary to X.509, a PGP certificate supports both *multiple* identities and signatures. Therefore, one can have one public key with multiple aliases attached to it. For example, a work identity using a corporate email address, and a private one.

Table 3.2: PGP certificate fields [Zimmermann, 2000, pg. 25]

Field	Description
Version Number	The used PGP version
Public Key	The public key of the certificate holder, as well as the corresponding algorithm
Personal Information	Details about the certificate holder, such as the name or email address
Digital Signature	The signature of the certificate structure. In PGP, certificates are at least signed by the certificate holder herself.
Validity Period	The certificate's start and expiration date

Each identity has a set of corresponding signatures vouching for its identity. There is at least one self-created signature per alias, sealing the contents of the certificate. However, a self-signed PGP certificate alone does not provide additional information about the person using the key. The intention of PGP is to have the certificate signed by multiple other participants. Therefore,

[2] https://www.torproject.org

a particular user is identified by the surrounding network of people who signed her certificate. Since one can have multiple identities with separate sets of signatures, people can sign either all associated identities or a subset of those.

3.3.2 The Web of Trust

The trust expressions, created by people signing each other's key, form a so-called *web of trust*. The web of trust is a directed graph, with the vertices being the participating people. The edges are trust expressions from one user in the key of another user. In PGP, trustworthiness is not the result of a decision by a remote, centralized TTP, but determined by the number of trust expressions of other users. A trust expression in PGP means that a user vouches for the authenticity of another user's key. Thus, it is a certification that the person behind a key is indeed who she claims to be. Such a trust in another user's key is expressed by signing the other user's public key with one's own private key, both of which need to be part of PGP's web of trust. All keys, including their signatures by other participating users, are uploaded to public key servers, to have the current set of trusting users visible to others. PGP encourages so-called *key signing parties*, where people meet in person, exchange their key fingerprints and sign each other's keys.

The web of trust does not necessarily have to be connected, meaning that usually not all participants can be reached by navigating the graph from a random starting vertex. Thus, there are typically multiple smaller webs of trust, as illustrated in Figure 3.5.

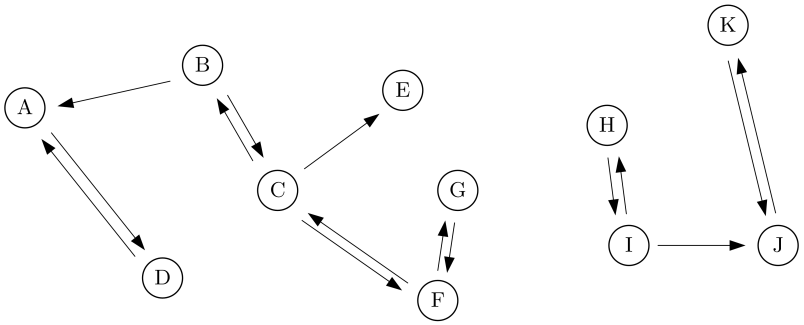


Figure 3.5: Sample webs of trust

3.3.3 Key Servers

PGP uses key servers to store people's certificates on centralized repositories. A key server is merely a convenience feature, since it is easier to keep a certificate with all its attached signatures up to date at a central location. Key servers themselves do not decide upon trustworthiness, they simply distribute keys to those asking for them.

It is common practice to upload a key to one or several key servers for people to access them with ease. Furthermore, once a key has been signed to express trust in the authenticity of a user, it is also common to update the key at the server, disseminating that trust expression to other users.

There are attempts to fully decentralize PGP, by replacing these key servers by distributed lookup mechanisms [Morselli et al., 2006].

3.3.4 Authenticity Assessment

Meeting in person to verify another person's key, such as suggested by key signing parties, is often impractical. Therefore, PGP supports a concept similar to CAs in the PKIX world, *trusted introducers*. As a CA does in PKIX, a trusted introducer in PGP is capable of deciding upon the trustworthiness of a particular user *for someone else*. A user who trusts a trusted introducer will also accept all keys signed by this trusted introducer. In PGP, every participant can be a trusted introducer, since everyone can sign other keys. However, no one is automatically accepted as being capable of deciding upon trustworthiness for another user. Each participant may act as a trusted introducer for any number of other trusting users, depending on how this participant is classified from the trusting user's viewpoint.

To explain how a trusted introducer is established, a look at both the following levels of trust and validity is required. In addition to the possibility of signing a key, one can also specify the level of trust in the key owner's ability to assert and certify the validity of other keys.

There are three levels of trust, besides *implicit trust*, which is the trust in the own key pair:

- Complete trust
- Marginal trust
- No trust (untrusted)

Furthermore, there are three levels of *validity* as well:

- Valid
- Marginally valid
- Invalid

Suppose a scenario with three PGP participants, Alice, Bob and Carol. If Alice wants Bob to act as trusted introducer for her, his key has to be *valid*. That is, it has to be either signed by Alice herself, or signed by another trusted introducer of Alice's. Alice can then set the level of trust of Bob's key such that he becomes a trusted introducer.

If set to *complete trust*, Bob will act as trusted introducer for Alice, regardless of any other signatures attached to Bob's certificate. If Alice decides to *marginally trust* Bob, there has to be at least one other signature attached to Bob's certificate which also expresses marginal trust for Bob to become Alice's trusted introducer. If Bob is a trusted introducer for Alice, and Bob signs Carol's key, then Carol gets indirectly trusted by Alice.

PGP supports manually setting *meta-introducers*, which is the equivalent of a root CA in a PKIX scenario. A meta-introducer can express its trust in other users to be trusted introducers. Thus, it resembles the root CA and subordinate CA construction.

3.3.5 Revocation

PGP allows every signer to revoke their signatures on other certificates. Moreover, both certificate holders as well as designated *revokers* are capable of revoking a certificate entirely.

In practice, however, revocation is hardly effective, due to the wide dissemination of certificates. A certificate owner might revoke her certificate, but one never knows where, possibly outdated, copies of the certificate reside. Even when spreading the revocation information across the most popular key servers, people might still rely on local copies of the certificate.

3.4 SPKI/SDSI

Simple Public Key Infrastructure (SPKI) was proposed by Ellison [2011; 1999] and merged later on with a concurrently emerged and similar project, Simple Distributed Security Infrastructure (SDSI), by Rivest and Lampson [2001; 1996]. The umbrella term which unifies both approaches remained SPKI. Despite the interesting concept, SPKI never made it out of the academic world. However, it is worth elaborating on the core concepts of SPKI, because it is distinguishable by both the fact that it emphasizes on *authorization* and on minimizing the associated *risk* when trusting another party.

SPKI is decentralized and makes use of so-called *name certificates* and *authorization certificates*. Each participant has at least one key pair and one or more self-chosen identifiers. Each key pair denotes a local namespace, linking the key pair and identifiers together. A participant can issue name certificates to be used within that local namespace.

To give an example, following the notation of [Clarke et al., 2001], consider Alice (A), Bob (B) and Carol (C). Alice might issue a certificate for *Bob*, including him in her local namespace, as defined by her key pair. This certificate, denoted by $K_A Bob \rightarrow K_B$, creates an alias in Alice's namespace, referencing Bob's public key using the name *Bob*. The key on the left hand side of the statement, K_A, followed by the local label *Bob*, denotes the fully-qualified name of Bob, as expressed in the namespace of Alice. The notation of having the key followed by a local label is a reference to both the namespace, as *uniquely identified* by the key, and the local label contained therein. The key K_B on the other hand, not followed by a label, is the key identifying Bob himself.

Furthermore, assume that Bob issued a name certificate for Carol, using her full name as alias in his local namespace, $K_B CarolJones \rightarrow K_C$. Alice can then include Carol in her local namespace, using whatever alias she desires, by having Bob act as an intermediate broker, for example $K_A Carol \rightarrow K_B CarolJones$. Contrary to the first two statements, K_A and K_B denote namespaces, since both are followed by local labels. Therefore, Alice's reference to *Carol* is just a link to Bob's entry *CarolJones*, which in turn directly links to the key of Carol. This sample setup leads to the namespaces summarized in Table 3.3.

The result name certificate *chain* can be traversed by resolving the individual namespace references. For example, starting at Alice, Carol is resolved by translating the local name *Carol* to Bob's *CarolJones*, leading to her key. Thus, these pointers to other's namespaces form a graph structure, similar to the web of trust.

Table 3.3: SPKI example namespaces

Namespace of Alice	Namespace of Bob
$K_A Bob \rightarrow K_B$	$K_B CarolJones \rightarrow K_C$
$K_A Carol \rightarrow K_B CarolJones$	

Besides the name certificate concept, there are authorization certificates. These certificates contain both source and destination as well as a delegation bit, validity information and their purpose. Therefore, SPKI incorporates the widely understood concept of trust being tightly coupled to a certain task, much like one trusts a mechanic for fixing a car, but not for heart surgery [Abdul-Rahman and Hailes, 1997; Blaze et al., 1999, 1996; Clarke et al., 2001; Yahalom et al., 1993].

For example, Alice could authorize Bob to sign a certain contract. Such a resulting authorization certificate is typically very limited in its validity period, such that potential risk is counteracted with fine-grained control of both purpose and duration. Multiple authorization certificates cumulate, with the intersection of all authorizations representing the actual permission. If one issues the permission to read and write to a certain FTP server and then reissues another authorization certificate allowing only read access, then the recipient will have read access only. Authorization certificates can be issued with a delegation indicator set, which allows the recipient to pass the grant to someone else.

Therefore, SPKI pre-emptively addresses the revocation problem by issuing only short-lived authorization certificates in the first place. CRLs are supported as well, next to means to limit an authorization certificate to only one use [Ellison et al., 1999].

3.5 DNSSEC

Domain Name System Security Extensions (DNSSEC) is a suite of extensions to the existing Domain Name System (DNS), defined in the RFCs 4033 [Arends et al., 2005a], 4034 [Arends et al., 2005c] and 4035 [Arends et al., 2005b].

The goal of DNSSEC is to provide origin authentication and data integrity of DNS data as well as authenticated denial of existence for non-present records. Its initial purpose was to protect users from forged DNS records, arbitrarily redirecting them to other, malicious hosts. Therefore, all DNS responses are signed by the ultimate DNS authority, residing at the top of the DNS hierarchy, when DNSSEC is used. DNS is responsible for name resolution and the DNSSEC extensions tie a domain name to a person or organization by means of their public key, authenticated at the root servers or one of their delegates. Thus, if comprehensively deployed, DNS root servers are the equivalent of root CAs in X.509, so called trust anchors. Intermediate DNS servers, who cache local information to keep traffic to the root servers low, play a similar role as subordinate CAs.

DNSSEC counteracts attack scenarios such as DNS Cache Poisoning, where intermediate DNS servers are tricked to cache a wrong mapping from domain name to Internet Protocol (IP) address. As DNS messages can be extended to also contain certificates in a separate CERT resource record [Joseffson, 2006] field, DNSSEC can be leveraged to deliver signed certificates. Therefore, it can actually replace X.509 in terms of its functionality.

One drawback of X.509, the ability for all trusted roots to sign any arbitrary domain, is also found in DNSSEC, except that the accepted roots are the DNS root servers. DNS, as X.509, is a strictly hierarchical system, thus some of the structural problems of X.509 reappear. The doubts alongside trusting an unknown third party remain. Moreover, if both domain name lookup and public key certification converge in one ultimate, trusted authority per top-level domain, then potentially even more harm is done if that authority fails or misbehaves.

One improvement of DNSSEC over X.509 is the binding of public keys to domain names. PKIX uses Distinguished Names, whereas DNSSEC implicitly binds the information to the domain name, since it is directly embedded into DNS. This binding is especially useful in conjunction with DNS-based Authentication of Named Entities (DANE), which aims to include public key information of applications into DNS records as well. For example, the TLS public key of a web service might be directly included into the DNS record, which allows for further cross-checks and thus tighter binding [Hoffman and Schlyter, 2012].

Revocation was added to DNSSEC at a later point, specified in RFC 5011 [StJohns, 2007]. It allows DNSSEC-aware to deal with key compromisation on DNS-record level. RFC 5011 introduces a revocation bit to DNSSEC records, which invalidates a trust anchor. Other resolvers, who previously set a now-revoked entry as trusted anchor, will no longer consider it upon chain verification. Thus, a DNSSEC chain ending at such a revoked point will no longer be considered valid. Revocation information is intended to be automatically updated between DNSSEC-aware hosts.

Meanwhile, DNSSEC is deployed for numerous Top Level Domains (TLDs), for example `.at`[3] or `.de`[4]. However, if DNSSEC is actually used depends on whether the DNS server configured at the user's supports it. It means that these TLDs are verified by a root server, but the chain of trust is not necessarily continued any further. At the time of writing, the adoption of DNSSEC for single host records is very limited. Moreover, most client-side implementations do not support DNSSEC validation or have it turned off by default.

3.6 Trust by Reputation

Trust and risk are strongly related [Das and Teng, 2004]. Trust as a perception is required for engagements when risk is at stake. In such a risky, uncertain situation, it seems natural to consult the wisdom of others before manoeuvring oneself into a dependent position [Abdul-Rahman and Hailes, 1997]. The idea to incorporate reputation into decision-making in such a situation is probably as old as mankind. Digital, especially on-line systems, made this explicit via reputation systems. Reputation systems appear in various forms, for example as user review sections in on-line shops, Q&A portals, recommender systems ("people who bought this article also bought [..]") and even more so in the context of social networks (e.g. "liking" something) [Jøsang et al., 2007].

The purpose of such tools is to increase confidence in whatever is advertised or presented. PGP is one system which exploits this in an authenticity context, by having other users express their trust in the authenticity of a certain public key. However, numerous other systems have emerged,

[3] See http://www.nic.at/service/technische_informationen/dnssec/
[4] See http://www.denic.de/domains/dnssec.html

especially in the context of social networks [Abdessalem et al., 2010; Artz and Gil, 2007; Golbeck, 2006]. Proposals range from deriving access control regulations using social network graphs [Carminati et al., 2009], automated trust evaluation by analyzing relationships [Ashri et al., 2005] to even predicting trust in social networks [Dubois et al., 2011]. From a security-related viewpoint, reputation systems are tried to be embedded into identity management scenarios [Jøsang, 2007; Steinbrecher et al., 2011; Zimmermann, 2000], where the personal identity is confirmed using their surrounding's reputation as a measure for authenticity. Sabater [2003] provides a comprehensive overview of computational trust and reputation models.

Research in reputation-based approaches to trust gained attention in recent years, especially in the context of the semantic web, with the goal of having autonomous agents reason about the trustworthiness of others. A central point in reputation-based systems is to derive a metric for trustworthiness. To achieve this, several different trust models have been classified, depending on how the recommendations or threshold values are computed: discrete trust models, probabilistic trust models, belief models and fuzzy models [Abdessalem et al., 2010; Jøsang, 2007]. Despite its interesting nature, we do not discuss this topic in further detail as it is not directly relevant to the proposed system.

One example close to STUNT, in terms of using a reputation-based approach to assess a host's trustworthiness, is Web of Trust[5]. Web of Trust introduces a reputation system for rating web sites on the Internet. The safety of a web site is expressed by a few predefined and individually rated categories, such as *trustworthiness* and *vendor reliability*. Every user can rate a web page, and get aggregated votings as a guideline about any page they are about to visit.

3.7 Summary

Table 3.4 recapitulates and compares the introduced mechanisms in terms of the discussed properties. Neither the list of existing approaches nor the summarized properties are claimed to be complete. It is merely a collection of relevant aspects when comparing other work to STUNT. All approaches come with inherent advantages and disadvantages, which were discussed briefly.

Table 3.4: Summary of existing approaches towards verifiable host authenticity

Property	PKIX	PGP	SPKI	DNSSEC
Structure	Originally strictly hierarchic, meanwhile loops possible	Graph	Graph	Tree, hierarchic
Trust Decision	TTP	User decision, depending on other's vouchings	Presence of user-issued authorization certificate	TTP
Naming Scope	Global	Global	Local	Global
Revocation	Blacklisting (CRL, OCSP)	Labeling, redistribution	Focus on tight constraints, also blacklisting	Blacklisting, revoked bit

[5] https://mywot.com

STUNT is another alternative to determine a host's authenticity in large-scale networks. It uses a method similar to PGP, with an underlying network of trust and pursuing a related reputation-based approach. In the following chapter STUNT will be discussed in detail.

Chapter 4

STUNT: a Simple, Transparent, User-centered Network of Trust

"Let us not look back in anger or forward in fear, but around in awareness."
– James Thurber

A Simple, Transparent, User-centered Network of Trust (STUNT) is another, different effort to mitigate the authenticity problem in an Internet context. STUNT in its current form has been published [Potzmader et al., 2013], and the following description of the system is an adaption from the original source.

4.1 Goals

STUNT is designed with a list of goals in mind, discussed in the following list:

New Viewpoint	Provide a fresh approach towards solving the authenticity problem in large networks, where it cannot be expected that people know each other personally.
Users are Key	Have users decide about trustworthiness. We believe that trust is tightly coupled to the individual, and that trust decisions cannot be delegated to a list of automatically trusted entities, shared by all users across the world. Instead, the list of trusted hosts shall be able to vary from user to user. Furthermore, it is a stated goal that the system shall raise awareness about the authenticity problem in general, and that there is no waterproof solution for it up to now.
Resilience and Decentralization	Try to punish misbehaviour in a self-regulating way, without the need for controlling instances.
Integral Revocation	Make revocation an integral part of the system design, and pursue a whitelisting validity approach.

We are well aware that replacing the current PKIX approach is a tremenduous task, and certainly not going to be happening in the near future. However, we think that it is time to think

about new forms of addressing the authenticity problem. Therefore, the main goal is to introduce one alternative approach, if only to get people to rethink the current state and work on better alternatives.

4.2 Introduction

Binding keys to persons or legal entities is non-trivial and, for now, never completely certain. A secure connection requires an authenticated endpoint, and this authentication is hard to do in an Internet context, where users and host operators typically don't know each other personally. STUNT is a new effort to have users authenticate endpoints, based on confidence expressions of other operators. These confidence expressions, called trust relationships, are set up by the host's operator, and can be established with any other STUNT-aware host. A trust relationship between two hosts means that both *host operators* express their confidence in that the other *host* is indeed operated by whom it claims to be. The SSL certificate, containing the public key of the operator, is thus bound to the operator herself by all instances who vouch for its authenticity. STUNT is decentralized, with no need for TTPs. Instead of having pre-defined lists of trusted vouchers, as is the case in PKIX, users themselves are put into the verification role. Users who assess a host will have to decide if the presented cluster of trust relationships is plausible, and if the vouchers are acceptable trustors, i.e. if their trust expression has value for oneself or not.

Trust in the context of STUNT describes the confidence, expressed by a host operator, that another host is indeed operated by whom it claims to be. It does not necessarily infer that the host or its operator has legitimate or positive intentions towards users. Therefore, it does *not* describe if the host operator per se is trustworthy. In STUNT, trust is *mutual* between service operators, which forces them to enter relationships where both parties are dependent on each other. Between clients and servers on the other hand, trust is only unidirectional. Users may be trusting in the authenticity of hosts, not vice versa. *Distrust* on the other hand might well be present in a STUNT context, but cannot be explicitly labeled as such within the system.

Figure 4.1 shows a simplistic sample of a STUNT trust network. Users may or may not trust hosts in an unidirectional way, depending on their decision, and hosts can mutually trust other hosts. Hosts need reasonable trust relationships to appear credible to users, and users assess a host's authenticity based on these trust relationships.

Much of the responsibility which currently resides at CAs is shifted towards individual host operators. A host's operator is responsible for verifying the identity of another operator before engaging in a trust relationship. A processing step in the trust relationship engagement protocol requires the operator to acknowledge the fingerprint of the other party's certificate, much like it is done in SSH, for example. Operators are encouraged to use a second channel to verify this fingerprint. Depending on the importance of such a trust relationship, this might be a simple phone call or email, up to a contractual agreement. Operators need to be careful about whom to trust, because they will be trusted by the same party vice versa, due to the mutuality of the trust relationships. With the number of overall available trust relationships being severely limited, each trust relationship contributes to one's own reputation in a non-negligible manner. Trust relationships are designed to be mutual, since engaging with a less trustworthy party is intended to backfire at the initiator.

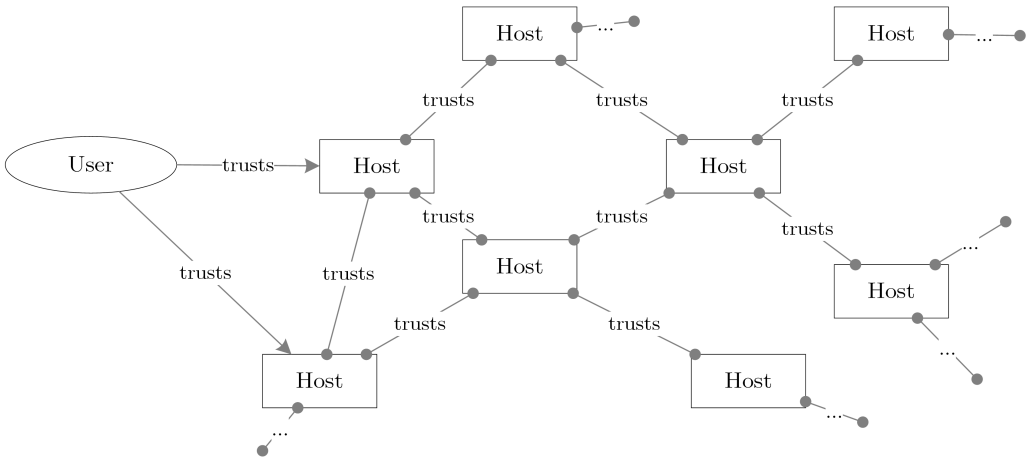


Figure 4.1: Trust relationships in STUNT

The number of available trust relationship *slots* is strictly limited. This limitation is achieved by having both involved parties compute a *proof of work* during the trust relationship establishment. The difficulty of the proof of work is dependent on the number of active trust relationships the *other* party has. Providing a proof of work is more difficult and thus expensive if engaging with a prominent host, which already has multiple active trust relationships. On the other hand, trust relationship engagements with new hosts is comparably cheaper. This asymmetry in effort models reputation in the system and acts as an incentive to include new hosts in the trust network. The less risk and more gain one has when engaging with an already-renowned host comes at a higher cost than a trust relationship engagement with a new, yet untrusted host. On the other hand, the latter option bears more risk and less gain. The difficulty of these required proofs of work is increased at a very fast rate, such that the problem is becoming computationally infeasible to solve if a target host has reached the maximum number of trust relationships. The number of overall available trust relationship slots is thus limited by the parametrization of the start difficulty and the rule to increase the difficulty per additional trust. The limitation of available trust relationships is in place for two purposes. First, to keep the trust network *immediately readable* for users, which is an effort to make the information as *explicit* as possible [Kirsh, 1990]. That is, users should grasp the directly surrounding network on first sight upon verification, without having to mentally process hundreds of nodes. Moreover, it gets very hard to find a seemingly unreasonable host in a network of hundreds. Second, trust relationships are intentionally limited to increase their *meaning*. With only, say, five available slots, each individual trust relationship has higher value than if there were a thousand available.

Jøsang [2011] distinguishes trust into *evaluation trust* and *decision trust*, where evaluation trust is the isolated, personal assessment of trustworthiness of the object in question. Decision trust on the other hand includes the risk factor. With a low possible loss, low evaluation trust might be sufficient to trust another party and thus become dependent on it. A high value at stake, however, requires high isolated evaluation trust. Figure 4.2 illustrates the relationship between decision trust and evaluation trust. STUNT follows this model by limiting the trust relationships. Since the value at stake is increased by severely limiting the available trust relationship slots, operators are forced to choose partners to whom they attributed high evaluation trust. If host operators do not carefully choose their partners, they may harm their own reputation.

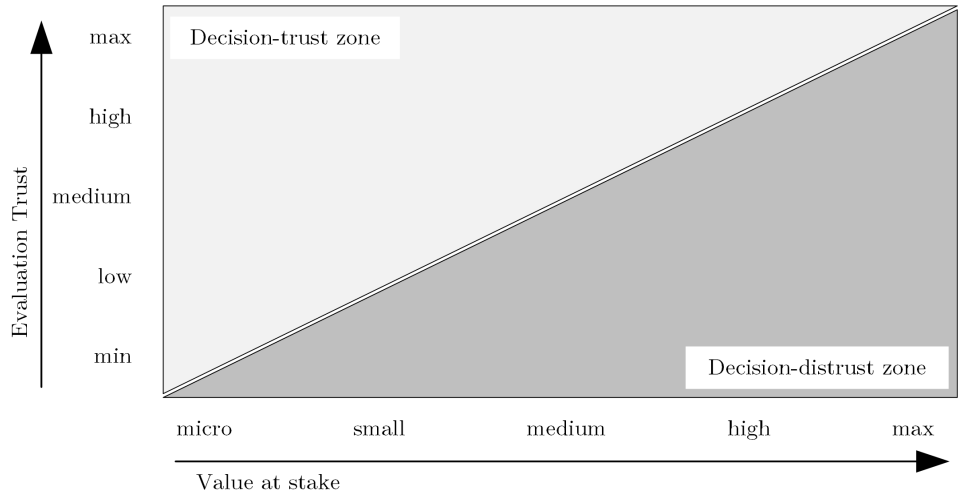


Figure 4.2: Relationship between evaluation trust and decision trust [Jøsang, 2011]

STUNT is about having users make decisions, instead of having some unknown infrastructure deciding for them. However, users certainly don't want to be annoyed with warnings or means to assess the authenticity of the host they want to visit all the time. Therefore, STUNT is designed for continuity. Trust relationships are intended to be long-term, because users who assess a host are able to store a local copy of the trust network and have it automatically verified on subsequent visits. The system ensures that, on subsequent visits, a host is indeed in the same state as it was assessed by the user. Deviations to this state will be notified, which is discussed in more detail in Section 4.3.7. The proof of work, which has to be computed upon the establishment of a new trust relationship, helps avoiding frequent trust relationship changes by host operators, since it is a costly, time-consuming task. Additionally, the value of trust relationships is increased, because one cannot replace it within seconds, but may have to wait days or even weeks until the replacement trust relationship is successfully established.

Each STUNT-capable host has to keep track of changes in the configuration of trust relationships with other hosts in an audit trail. The costly proof of work is bound to a certain state in the other host's audit trail using a hashchain over the entries present at the time of the engagement. As a result, clients can assess whether the difficulty of this task was adequate, given the number of active trusts of the other party at that state. To ensure the freshness of an active relationship, periodic updates are sent in intervals chosen by the parties themselves during their risk assessment. Clients will perform whitelist-based revocation checking. Only those hosts where both the proof of work verifies correctly and the relationship is proven to be recent, for all presented trust relationships, are considered valid. Therefore, revoking an existing relationship means to stop sending further updates.

To operate STUNT, an additional service is required on servers, and a browser module capable of performing verification and showing a network explorer to users is required on clients. Furthermore, a control interface is required on the server, to let host operators engage in new trust relationships or revoke existing ones.

In subsequent sections, the technical aspects of STUNT will be discussed in more detail, followed by an informal security analysis, which discusses possible attack vectors and their mitigation.

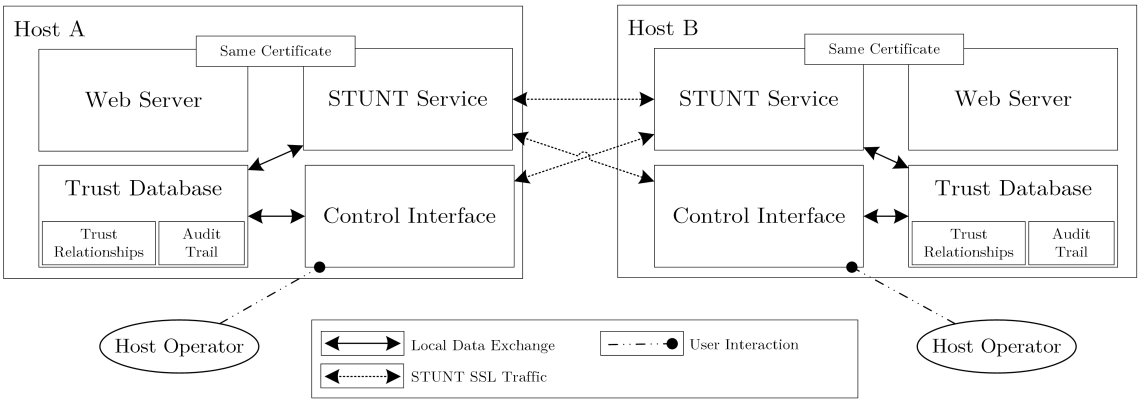


Figure 4.3: STUNT components and client-server communication flow

4.3 Architecture

STUNT can be operated in standalone mode or in conjunction with the existing PKIX infrastructure. The latter approach, called compatibility mode, supports having both users verify a host's authenticity using the network of trust and the CA-based model of verification at the client's. However, if operated in compatibility mode, STUNT is required to use the existing X.509 certificates, which is otherwise not necessarily the case. In standalone mode, all host certificates are self-signed. In compatibility mode, however, they may be signed by a Certificate Authority.

In this thesis, we focus on compatibility mode in this section, since introducing STUNT to a wide range of users is considered only possible in an incremental, add-on manner. Furthermore, we discuss a web context, where hosts are web servers offering HTTPS connectivity and clients are web browsers. STUNT is not limited to this application, but other scenarios can be operated analogously and are thus not discussed separately.

4.3.1 Components

The proposed system introduces an additional service on servers as well as client-side code to browse the STUNT network and verify its validity. Regardless of the mode of operation, the STUNT service has the *same certificate* as the web server, which ties both instances together and avoids rogue STUNT services, since clients can compare whether the certificates match.

The additional server component is needed to handle network-related operations, such as establishing or removing trust relationships, delivering the current status and keeping the trust relationship current. It is merely a service on top of a small local database that stores the established trust relationships as well as the local audit trail. Furthermore, it is responsible for computing the proof of work and keeping revocation information up to date. Additionally, there is need for a control interface for host operators, where they can opt to establish a new trust relationship, revoke an existing one, or simply view the status of established trust relationships. On the client, a different, local database is maintained, which acts as the local trust base and thus remembers the hosts users have already labeled as trusted.

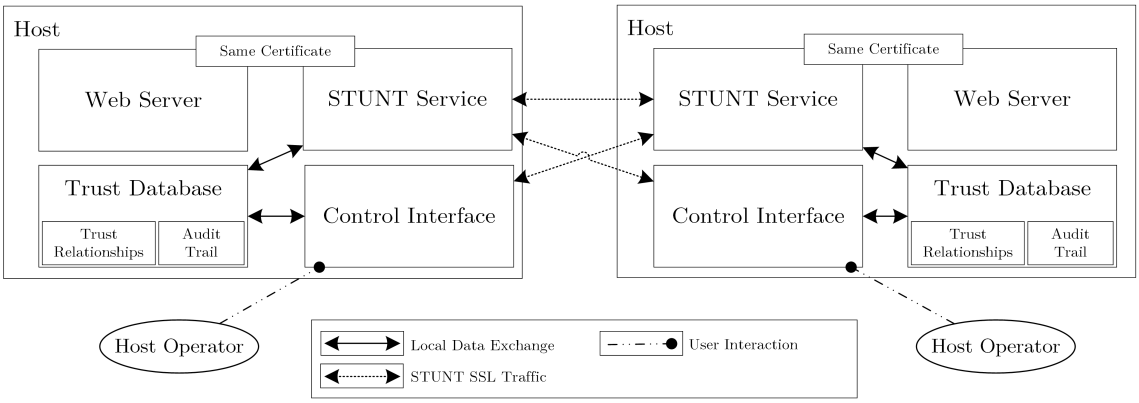


Figure 4.4: STUNT components and server-server communication flow

Figure 4.3 illustrates both the client and server components and how they interact. A user's browser, pointed at some destination, will query both the web server for the web site's content and the STUNT service for information about the surrounding trust network. The STUNT server component will then deliver information about any directly trusting nodes, if there are any trust relationships established. This information is verified at the client. If the user visited and trusted this particular host beforehand, the given information is compared against the previously stored data. If the host is being visited the first time, the user is notified and asked to verify the trust network. This notification may happen actively, by having a popup interrupt the user, or passively, by a simple indication stating that it might be good to assess this host's trust network. The regular HTTPS data flow is not affected in any way, except that warnings dealing with self-signed certificates might no longer be relevant at client-side.

Between servers, the communication flow is as depicted in Figure 4.4. If a host operator decides to establish a new trust relationship, the control interface is used to engage a new establishment with another host. It will trigger the other host's STUNT service, which in turn adds a pending request to the database. The control interface is monitoring the database, and notifies the other host operator of this new pending request. A new trust relationship can only be established once both parties acknowledged each other and entered the required risk interval. More details on the establishment of a trust relationship can be found in Section 4.3.6. Periodic, revocation-related updates as well as parts of the establishment protocol that do not require user interaction are handled by direct communication between the two STUNT services.

4.3.2 Audit Trail

STUNT-aware servers are forced to keep track of their trust relationship activities in an audit trail that is directly embedded into the system. This audit trail is machine-readable and keeps track of current and former trust engagements. It is merely an append-only list of the actions *initial setup*, *trust engagement attempt*, *trust engagement success*, *trust engagement rejected* and *trust engagement revoked*, alongside the corresponding timestamp and involved host, if any.

Both involved parties log attempts, successful creations, rejection and revocation events, using their local system time and the other involved host, respectively. The audit trail allows clients to verify at a later point whether a presented trust relationship was indeed creating in a legitimate

way. The trust relationship is bound to the state of the audit trail at the time of its creation. This is achieved by containing a hash chain over all audit trail entries available at the time. This hash chain prevents later insertions to the auditrail, since the trust relationship will then no longer verify. Since all additions and removals of trust relationships are recorded, clients are able to recompute the number of active trust relationships at the state defined by the hash chain. More details regarding the workings of this verification will be discussed in Section 4.3.7.

The possible entry types have the following function:

Initial Setup	A simple entry indicating the setup of the audit trail.
Trust Relationship Attempt	An entry recording whenever a host attempts to engage in a trust relationship with another host; to be appended by both parties.
Trust Relationship Success	An entry confirming a successful trust relationship establishment.
Trust Relationship Rejected	Records that the party receiving such an engagement offer denied the request.
Trust Relationship Revoked	A message indicating that a party just nullified an existing trust relationship.

The system-wide rule for audit trails is that for each attempted trust establishment, a subsequent reject or success message has to follow, before another attempt, either originating from this host's operator or targeted at this host's operator, is possible. Attempts which interleave a currently processed relationship attempt will be auto-denied. Pending requests that are not answered within a defined timeframe are auto-denied as well to prevent Denial-of-Service (DoS) scenarios.

4.3.3 Keys

STUNT requires two key pairs, the SSL key pair for the SSL connection and the STUNT key pair for signing elements in the protocol introduced in Section 4.3.6. If operated in standalone mode, a host's SSL certificate will be self-signed, because its authenticity is solely defined by the trust relationships. Regardless of the mode of operation, the STUNT signing key pair and the SSL key pair have to be strongly bound together.

To achieve this binding, we include the public STUNT signing key in the SSL certificate by means of a custom certificate extension, which allows seamless distribution of the public key and separates key usage between SSL and STUNT signatures. The STUNT key pair's sole purpose is to sign messages exchanged during the STUNT protocols and may not be used for a different purpose.

Figure 4.5 shows the structure of such an adapted certificate, where the public STUNT key is included and bound to the remainder of the certificate by its signature. This separate key comes at the cost that an existing certificate is not compatible to STUNT and will have to be re-issued. Moreover, if a CA additionally vouches for the authenticity of the host, the CA would have to agree to including the STUNT key into the certificate.

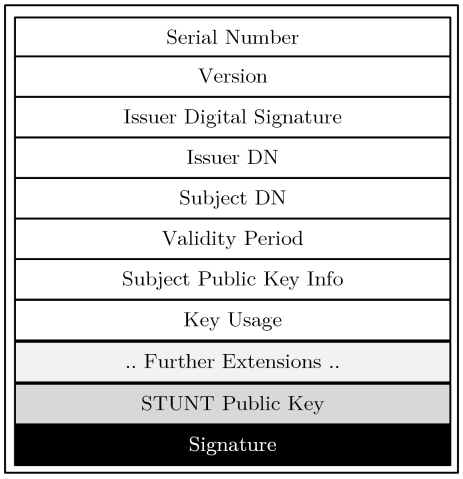


Figure 4.5: STUNT-extended X.509 certificate structure

4.3.4 Token of Trust

To seal an established trust relationship, two so-called *tokens of trust* are computed, one per party, and then exchanged. A token of trust binds a proof of work, the state in the audit trail, both parties' certificates and a freshness proof together, by concatenation and signing the whole content.

Suppose an operator of a host B accepts a request sent by the operator of host A. The unidirectional token of trust, as is then sent from A to B, is defined below. Note that the concrete protocol that needs to be executed for a successful trust relationship establishment is shown in Section 4.3.6.

$$\text{token of trust}_{AB} = (\left\{\begin{array}{l} H(\text{cert}_A \mid\mid \text{cert}_B \mid\mid \text{challenge} \mid\mid \text{padding}), \\ \text{hash chain}(\text{audit trail}_B), \\ \text{challenge}, \\ \text{padding}, \\ \text{update interval} \end{array}\right\}, \sigma_A(tot_{AB}))$$

Here, $\sigma_A(tot_{AB})$ refers to the signature over the structure denoted by curly brackets, signed by host A using the corresponding STUNT signature key. cert_X refers to the certificate of host X. H is a standardized, collision- and preimage-resistant hash function, for example SHA-256.

The proof of work is the digest generated by H. Computing a token of trust is a computationally expensive task of increasing difficulty. STUNT uses proofs of work as depicted in Figure 4.6, where a prefix containing both party's certificates as well as a cryptographic nonce, provided by B as freshness proof, has to be part of the preimage. The remainder of the prefix is a random number freely chosen by A, which is the padding required to get the desired output hash. A has to find a random padding, such that the resulting digest has n leading zeros, where n is dependent on B's number of currently active trust relationships. The difficulty n is adjusted by a predefined rule, to ensure that the problem's hardness is increased and thus the amount of maximum available trust relationships is limited. Such a random padding is hard to find. On the other hand, once such an expensive proof of work has been computed, it is very cheap to verify, since it requires only a plain

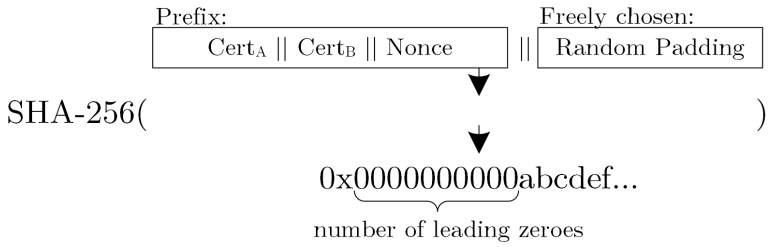


Figure 4.6: Proof of Work illustration

hash computation from the given input and an inspection of the output hash.

The token is bound to the other host's current state by including a hash chain over its current audit trail state, denoted by *hash chain(audit trail$_B$)*. The generated token of trust consists of the proof of work, the hash chain over the audit trail, the used padding, the originally given challenge and the predefined update interval, as well as a signature over these fields. The information in the token of trust allows clients to verify the correctness of the claimed relationship. Challenge and padding together allow clients to recompute the proof of work to determine whether the problem was adequately difficult upon the trust relationship establishment.

A token of trust seals a unidirectional trust relationship from host A to host B, provides a proof for adequate computational workload during its creation and is bound to the state of the other host's audit trail when it was created.

4.3.5 Revocation

STUNT uses an active, whitelisting- and risk-based revocation scheme. For a trust relationship to be valid, both a valid *token of trust* and a fresh *trust commitment* is required. Trust commitments are messages sent to all directly trusted hosts in the initially negotiated update interval t. A trust commitment, as sent from A to B, looks as follows:

$$\text{trust commitment}_{AB} = \sigma_A(\sigma_A(\text{token of trust}_{AB}), \text{timestamp})$$

It is thus merely a message containing a timestamp of when the message was created, the signature of the previously sent token of trust, and a signature over the message itself. When receiving such a message, hosts store the latest commitment message from the originating host and send it to clients upon request. For the trust relationship to be considered valid, the latest trust commitment needs to be no older than t.

4.3.6 Trust Relationship Establishment

To actually establish a trust relationship between two hosts, the protocol as shown in Figure 4.7 has to be executed. This protocol is merely the technical execution of a strategically important task between the two companies. At a certain level of importance, this may well involve setting

up contractual agreements, which include the hash digests of the public keys of the hosts involved as well as the chosen update intervals.

Much of the responsibility of common certificate authorities is shifted to host operators. Instead of having a CA verify the authenticity of a requester, it is now the operator's duty to do so. The backfiring principle acts as an incentive to support careful choices, but the user interface additionally enforces certain checks using a different channel. Before an operator can establish a trust relationship, it is necessary to specify the target host's public key hash digest, as well as the negotiated update interval for trust commitments. If the target host's certificate fingerprint matches, a request is sent. The recipient's STUNT service will notify the host operator of incoming requests. The protocol may as well incorporate an auto-denial mode, with only a small window of opportunity in which the service actually prompts the administrator to make a decision, and otherwise automatically denies requests. This mechanism can be put in place to keep the time span of being unable to process other requests short. The actual decision of whether or not to accept this request is intended to be made before the execution of the whole protocol, using the required second channel. A recipient who is notified of such a request has to respond manually by accepting or denying it. If accepted, the recipient has to enter the initiator's certificate fingerprint, as well as the update interval.

The protocol itself is not secured against Man-in-the-Middle attacks. Both parties have to ensure the other's authenticity. Establishing such a relationship is considered a rare and business-critical event. Operators are thus encouraged to exchange the certificate fingerprints using another channel, e.g., offline, when setting up a service-level agreement. Still, the protocol needs to be run on a SSL-backed connection to avoid message manipulation or eavesdropping by third parties, using the previously exchanged fingerprints to ensure authenticity.

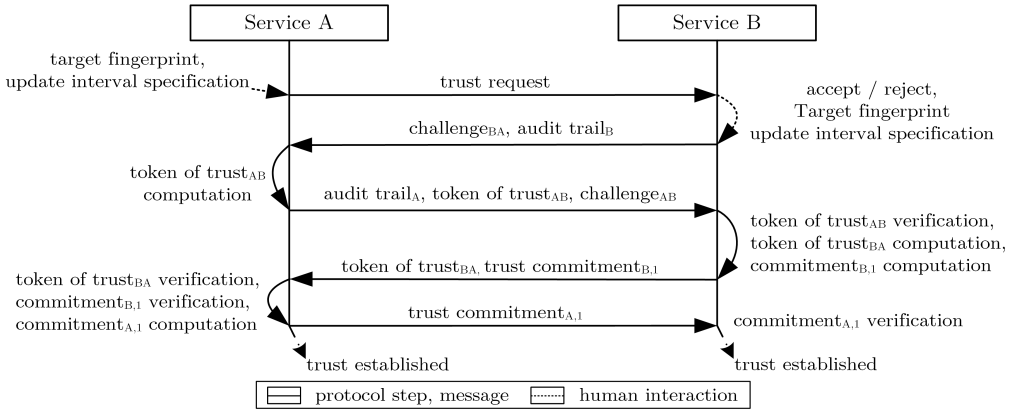


Figure 4.7: The STUNT trust relationship establishment protocol

Suppose the operator of service B accepts the request, which is only possible if and only if the window of opportunity is open, there is no second pending request, and the host is not already trusted. In case of acceptance, A computes a token of trust using B's certificate, audit trail and given challenge. The resulting token is sent to B, alongside its own current list of audit trail entries and a randomly chosen challenge. Subsequently, another token has to be computed by B and returned to A. Both sides' tokens of trust together are the expression of trust between the hosts' operators. Upon establishment, a token of trust is verified in the same way as in the client-side verification procedure for new hosts, described in Section 4.3.7. Additionally, however,

there is need to verify whether the included hash chain over the audit trail of a given token is indeed a chain over the whole, current audit trail. Once both parties exchanged their first trust commitment, the trust relationship is considered active by clients. If, at a later point, a trust commitment is not received within the defined update interval, the trust relationship is considered revoked.

4.3.7 Verification

Client-side verification differs depending on whether the client already considers the host in question as trusted or not.

New Hosts Listing 1 shows the verification procedure of newly visited hosts. For new hosts, the first thing to do is to fetch information from the host in question. That is, the host's certificate and its audit trail, denoted by `GET_CERTIFICATE_AUDIT_TRAIL(target)`, and the information about all directly trusting hosts, referred to as neighbors, denoted by `GET_NEIGHBORS(target)`. The certificates from both the STUNT-service and the actual service to be used, e.g. the web server, are checked for equality beforehand, which is omitted in Listing 1. Then, the verifier has to fetch all neighbor certificates, denoted by `FETCH_CERTIFICATE(target)`, to be able to verify the token of trust's signature as returned from the host in question, if the corresponding trust commitment is not outdated. Signature verification is done by `VERIFY_TOKEN_SIG(token_of_trust,token_signature,public_key)`, and the check for an outdated commitment is done in `RECENT(timestamp,interval)`, by comparing the trust commitment's creation date to the local time, using the token of trust's interval as reference.

Algorithm 1 VERIFY_NEIGHBORS(host)

```
host_cert, audit_trail ← GET_CERTIFICATE_AUDIT_TRAIL(host)
neighbor_info ← GET_NEIGHBORS(host)
for all neighbor in neighbor_info do
    tt ← neighbor.trust_token
    neighbor_cert ← FETCH_CERTIFICATE(neighbor.IP)
    tc ← neighbor.trust_commitment
    if not RECENT(tc.timestamp, tt.interval) then
        return false // outdated commitment
    end if
    if not VERIFY_TOKEN_SIG(tt, neighbor.token_signature, neighbor_cert.kpub) then
        return false // invalid token signature
    end if
    trusts ← RECOUNT_ACTIVE_TRUSTS(audit_trail, tt.hashchain)
    if trusts == -1 then
        return false // invalid hash chain
    end if
    if not tt.proof_of_work == H(neighbor_cert, host_cert, tt.random, tt.challenge) then
        return false // invalid proof of work
    end if
    if LEADING_ZEROS(tt.proof_of_work) ¡ n then
        return false // inadequate problem hardness
    end if
    if not tc.token_signature == neighbor.token_signature then
        return false // commitment for wrong token
    end if
    if not VERIFY_COMMITMENT(tc, neighbor_cert.kpub) then
        return false // invalid commitment signature
    end if
end for
```

Afterwards, element-wise hash computation and concatenation is performed on the given audit trail entries until the hash chain result contained in the token of trust is met, done in `RECOUNT_ACTIVE_`

`TRUSTS(audit_trail,hashchain)`. If it is never met, either the audit trail was modified or an invalid hash chain was given, in which cases `RECOUNT_ACTIVE_TRUSTS()` returns -1. Otherwise, one is able to re-count the number of active trusts until this very state, because all successful additions as well as all revocations are recorded in the audit trail. If one party does omit relevant audit trail entries, it will not be able to add further trust relationships, since the verification procedure will fail, either at the client or at other hosts, when attempting to establish a new trust relationship.

With that information, the verifier can recompute the proof of work using both involved certificates, as well as the given random padding and challenge. The number of the resulting digest's leading zeros is then compared in `LEADING_ZEROS(proof_of_work)` against the number of active trusts at the time of trust establishment, n, to verify whether an adequately hard problem was solved during the token of trust creation. The trust commitment is then verified by checking the signature and comparing the token signatures.

The user is only presented with the trust relationships if these verifications succeed. Otherwise, the client will abort prematurely and issue a warning. The system's purpose is thus to ensure that the presented information was indeed created at the neighbor's will and is not entirely made up. Whether or not users decide that the presented cluster of nodes is reasonable and to be trusted is decoupled from the system and might differ from one user to another.

Verification of these yet unvisited hosts consists of many steps. Nevertheless, the individual steps are computationally cheap and the number of maximum active trusts is bound by the proof of work complexity to some intentionally low number, such that the total workload is considered negligible. Section 5.2 contains a sample analysis of verification runtimes on a low-end device to support this claim.

Subsequent visits Assessing whether or not a specific newly visited host will be added to one's personal trust base is considered a rare event, once the initial trust base is established and contains typical, regularly visited hosts. In contrast, ensuring that subsequent visits are indeed addressing the very same host might occur much more frequently.

Adding a host to the personal trust base means to store its certificate, as well as all given tokens of trust and commitment timestamps with the corresponding neighbors and neighbor certificates in a local database. Assuming that the user revisits a previously added host and that the last stored timestamp is outdated, then the client fetches all recent commitments from the server. For each commitment, the client checks whether it originates from a previously stored neighbor, verifies the commitment signature, checks if the timestamp is recent and whether or not the token signature matches the stored one. Thus, on subsequent visits it is not necessary to connect to any neighbors. There are basically four cases that might occur in this verification:

- The host's state is exactly as it was upon accepting it. That is, all the given information matches the local database. In that case, clients accept the host without any further user interaction.

- The host established one or more new trusts with other hosts. Our approach follows the logic that having more trusts expressed by others is a good thing. Therefore, the user is not disturbed. Instead, a small visual indication will be shown, with the optional possibility to

re-assess the new trust network.

- The host lost one or more of the trusted hosts since it was labeled as trusted by the client. That is, the host in question is unable to provide a recent trust commitment of a trusting host. This is critical. For example, due to losing the trust relationships because of misbehavior. Hence, the client should interfere and warn the user with details about the formerly trusting nodes and why this might be a problem.

- One or more of the involved hosts changed their certificate. This means that the stored information is invalidated and the host(s) have to be re-assessed.

4.3.8 Limiting Trust Relationships

The number of active trust relationships is intentionally limited to a very small number. The reason for employing a cost function instead of limiting the number of trusts to a domain-wide value, such as five hosts for example, is to avoid fast switching of trust relationships and thus underline the long-term nature of these trusts.

The limitation per se serves the two purposes of keeping such a trust relationship a scarce and thus valuable resource, and to allow users to comprehend the whole cluster of directly connected hosts on first sight. Therefore, the idea is to limit these direct connections to a value as low as about five hosts. Achieving the desired number of maximum hosts is a matter of parametrizing the initial offset n_0, which defines how many leading zeros in the proof of work are required for the first trust relationship. Furthermore, the rule to increase the difficulty has to be specified. This rule defines the pace at which the difficulty grows, depending on the target node's reputation.

The proof of work parametrization is flexible and thus scalable. By altering the starting offset and the rule for n, varying difficulties can be achieved. To give an example, we assume a hash function with a preimage resistance strength of $2^{\text{digest length}}$, such as the SHA family [Dang et al., 2012]. Furthermore, we assume a sample device, capable of performing 50 million hashes per second. To delimit the maximum active trust relationships to five hosts, one example parametrization might look as depicted in Figure 4.8. The rule to increase the difficulty in this example is set as follows:

$$\begin{aligned} n_0 &= 36 \text{ and} \\ n_{i+1} &= n_i + i + 1 \end{aligned}$$

Thus, the difficulty is increased dependent on an initial offset n_0 and the number of already-active trust relationships. The highlighted data points denote the required leading zeroes for five trusts. Using the assumptions above, the proof of work computation would need up to 0.4, 0.8 and 3.1 hours for the first three trust relationships, up to one day for the fourth one and up to 16 days[1] for the fifth one. Since the complexity is increased with the number of already-active trusts, a sixth trust relationship would then already require a proof of work that takes up to 521 days to compute, which is far from reasonable from an economic point of view.

[1] These are upper bars, and one can always be lucky. However, even in the first case with $n_0 = 36$, given all possible output hashes, the probability to hit a correct hash is about $1.46 \cdot 10^{-11}$. For a 50% chance of getting a valid hash, half the time is needed.

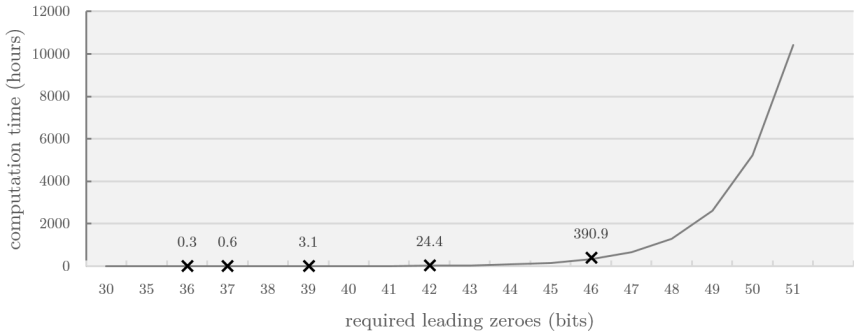


Figure 4.8: Proof of work sample parametrization

Delegating the task of computing a proof of work to a high-performance cloud computing instance will certainly decrease the computation times, but comes at a cost. A current Amazon EC2[2] top-end computation machine, a *Cluster GPU Quadruple Extra Large* instance, is specified as shown in Table 4.1.

Table 4.1: Amazon EC2 Cluster GPU Quadruple Extra Large instance specification

Component	Specification
CPU	33.5 ECUs, 64 bit
Memory	22 GiB
GPU	2x Nvidia Tesla "Fermi" M2050
Storage	1690 GiB
Networking	10 GBit Ethernet

According to Amazon, one Amazon EC2 Compute Unit (ECU) is comparable to a 1.0-1.2 GHz 2007 Opteron CPU[3]. Currently, such a machine, operated as a *spot instance*, which is the cheapest option that has no dedicated hardware, but instead consumes otherwise unused capacities, costs about \$0.2 per hour[4]. According to Oprisan [2013], such a system, when tested with an optimized SHA-256 implementation utilizing the GPUs, evaluates to 106.18 Megahashes per second. Therefore, it takes about half of the times depicted in Figure 4.8 to compute the proofs of work. In terms of money, using this configuration and the previous example, this would evaluate to \$0.03, \$0.06, \$0.31, \$2.44 and \$39.09, respectively. Computing a sixth trust relationship would cost up to \$1251 and take up to 260 days to compute. This finding of a fitting pattern can be efficiently parallelized. Therefore, significantly faster results can be achieved with higher investments.

Increasing computational power leads to faster solutions of the proof of work tasks. This might mean that, say, in ten years from now, a sixth trust relationship is viable. This might be a desired property, if the system is allowed to grow in terms of its allowed links at a very slow pace. If not, however, the token of trust can be equipped with a simple version indicator. Such a version indicator can later be used to restrict tokens created after a certain timestamp to a higher version to be accepted, with a new difficulty configuration attached to that version.

Note that the number of required zeros is reduced again as previously increased when revoking a

[2] https://aws.amazon.com/en/ec2/
[3] See https://aws.amazon.com/en/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it
[4] See https://aws.amazon.com/en/ec2/pricing/, prices are highly volatile and dependent on the server location

trust relationship, because there is a corresponding revocation entry required, which is visible to clients.

4.3.9 Trust Communities

Trust tokens and thus the trust relationships are bound to a certain audit trail by the hash chain. While operators cannot alter an audit trail later on, nothing prevents them from starting a second one from scratch. Maintaining several audit trails lets operators increase and *categorize* their trust relationships. For example, a company might operate one category containing reference clients and another category containing suppliers, each limited to n hosts, where n is computationally bound by the proof of work computation complexity. From a user's point of view, these categories could be specially highlighted in the browser, or there might be filters in place, allowing users to explore only a certain category.

4.4 Security Considerations

The following informal security discussion shows attack vectors on STUNT and follows the adversary model by Dolev and Yao [1981]. According to the model, an adversary is assumed to have full network control, but no host itself is compromised. Several possible vectors for mounting Man-in-the-Middle (MitM) attacks exist in this scenario, which are now discussed briefly. A successful MitM attack allows adversaries to read and modify exchanged messages without being detected by either the client or the server involved. Impersonating the target host without relaying the messages to the actual target is another possible approach, but treated as the same in this analysis, as there is little difference in how this attack is mounted. In this analysis, we consider only server impersonation, as STUNT is about authenticating server entities. Thus, an attacker has no gain in terms of appearing trustworthy in STUNT when impersonating a client. Since full network control is assumed, an attacker can impersonate a server from both the clients' and other servers' perspective. For example, impersonating a server from a client's viewpoint is possible on public wifi hotspots. However, it is also thinkable in a larger context, that entire companies want to monitor their employees' secure communication or countries want to eavesdrop on their citizens.

It is important to remember that both a server's STUNT- and web server use the same certificate for authenticating the communication. Therefore, an adversary redirecting only the STUNT requests to some spoofed service will be noticed, as the certificates will no longer match. Hence, adversaries are forced to spoof both services in conjunction. Furthermore, host operators are encouraged to take trust relationship decisions seriously. There is only a limited number of possible mutual trust relationships, and establishing them is computationally expensive and thus costly. Moreover, establishing a trust relationship needs an agreement using a second communication channel, for example a contract.

We identified and named two possible attacks, referred to as *replacing* and *remodeling*, which will be discussed below.

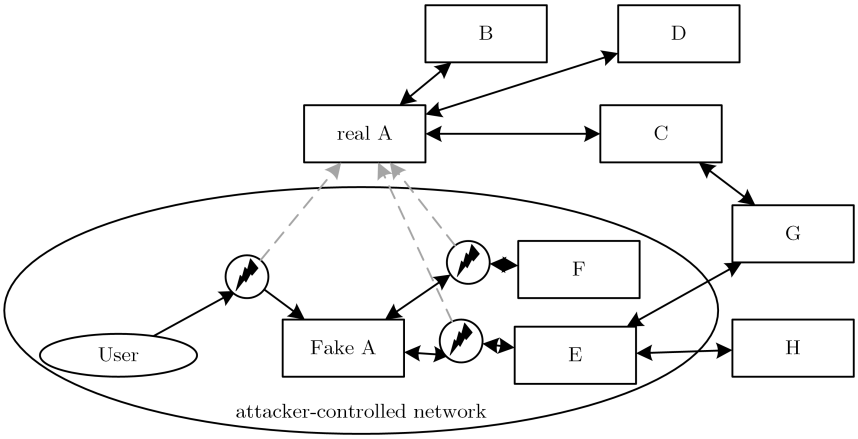


Figure 4.9: Schematic of the replacing attack

4.4.1 Replacing

Description Full network control allows an attacker to impersonate a server in the network. An additional server with the same DNS name as an existing one can be introduced, and all client requests to the real host can be redirected to the impersonated one. The impersonated server can then be used to lure other operators into establishing trust relationships with it. Given that other operators agree on these trust relationships, an adversary might be able to attach a malicious node to the otherwise legitimate trust relationship network. Due to the same DNS name, the maliciously introduced server *replaces* the original one. Figure 4.9 shows a *replacing* scenario, where a host A is being replaced. The attacker is assumed to control both the links of the user's client, and the hosts E and F, although increased network control would make little difference. The attacker introduces a fake host A, and redirects trust relationship requests from E and F to the fake A. If the operators of E and F accept the trust relationship with the fake A, believing that it is the real A, then this malicious host is suddenly attached to the otherwise legit STUNT network of trust. From a the user's perspective, the fake A might seem real, since it is connected to other renowned hosts, having legitimate vouchers on their own.

There is also a weaker form of *replacing*, which works having only the server's link under control, whereas the client's connection remains untouched. However, it requires attackers to choose a new, available DNS name and can thus only introduce new, seemingly trustworthy hosts.

Discussion The server operators are responsible for authenticating other servers, otherwise it is theoretically possible for an adversary to gain trust relationships. The system requires operators to take special care whom they trust. The expensive creation and mutual nature of trust relationships emphasize the importance of this decision. Whenever a host operator decides to trust another host, she will receive the trust vice versa. There is only a limited amount of incoming trust relationships, and adding an untrusted host backfires, because it reduces the likeliness of being accredited as trustworthy by users. Since adding trusts is expected to be a rare event, operators are asked to exchange the expected fingerprints on a second channel, which counters rogue trust relationships. If the attacker manages to set up such a seemingly trustworthy host with the same DNS name, there would still be a warning if the attacked user has visited the real host before. A warning is

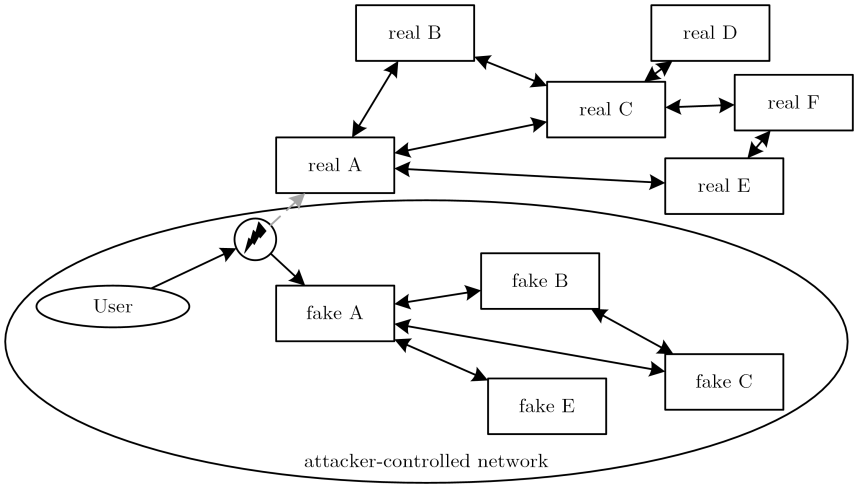


Figure 4.10: Schematic of the remodeling attack

issued, because the presented and the previously stored certificate from the original server would not match any more. Furthermore, we do not expect such trust relationships to last for long, since the fraud is likely to be detected soon by the tricked host operators. Aware users might detect this as well, when they end up at the real host when deeply exploring the trust network. For instance, the user depicted in Figure 4.9 could browse from the fake A to E, G, C and then see the real A popping up in the network explorer.

4.4.2 Remodeling

Description Instead of getting other operators to establish a desired trust relationship, an attacker might as well fake a network of a trusted server by adding a number of self-established hosts. We call this *remodeling*, since the attacker tries to copy an existing network of trust, or to create a similar one. This attack requires only client-side networking control. Figure 4.10 shows a sample of such a *remodeling* attack. Here, the user's requests for the real host A are redirected to a fake host A created by the adversary. In addition to faking A, the attacker has created fake hosts for all directly trusted nodes of the real A. Therefore, the trust network seems the same to a user who does not browse deeper than the first level of trusting hosts.

Discussion Performing this attack is a lengthy and expensive task due to the required proofs of work. Furthermore, its success is highly dependent on the user's reaction, because it can not be known beforehand how much effort the user puts into investigating the presented network. A *remodeled* trust network is assumed to be very small, because each additional host to fake is computationally expensive for the adversary. Since the user can browse the network arbitrarily deeply, the deception will eventually come to light. We expect STUNT networks of trust to be large in terms of their number of vertices. A remodeled network with a limited amount of nodes might be suspicious to users. In the large-scale case of companies and countries, such fake networks get more realistic. However, their use is limited, which renders the expense/gain ratio very low. They will not last for long, because people typically visit hosts like their on-line banking not just once and from different places. Their clients store the corresponding certificates and will thus issue

warnings as soon as a subsequent visit yields different fingerprints. More people being affected means more gain for the attacker, but the faster it will be detected.

Furthermore, to render this vector almost completely unusable, the system can be used in conjunction with Perspectives [Wendlandt et al., 2008], having notaries on various places in the world monitor certificate fingerprints. An additional variant would be to provide a tool which allows easy comparison of the presented trust network on a desktop client to the one that is presented on the mobile device, which usually uses a connection provided by an independent carrier.

Chapter 5

Welcome to STUNTLand, a STUNT Lab Environment

"The irony of the process of thought control: the more energy you put into trying to control your ideas and what you think about, the more your ideas end up controlling you."
– Nassim Nicholas Taleb

We implemented a prototype of STUNT in Java and tested it in a virtual environment to show that it is indeed implementable and to get a feeling how it performs in a small setting. To test STUNT, all components had to be implemented, including the server service, the control interface, and the web browser. Due to understandable security restrictions regarding working around a CA-driven infrastructure in current web browser add-on Software Development Kits (SDKs), we implemented a standalone browser using the web engine of `JavaFX`[1], instead of developing an add-on for a common browser.

5.1 Component Design

In this section, we discuss the necessary STUNT components in more detail, to give an overview about the prototype implementation.

STUNT server The STUNT server is a service to be run on each STUNT-capable host. It is mostly an interface to the local database, which contains the trust relationship information as well as the audit trail. In addition, it performs the logic required to establish trust relationships and update trust commitments. Therefore, it incorporates the proof of work computation, token signing, and message exchange. The local databases, both the one on the client and the one on the server, were implemented using `SQLite`[2] as backend Database Management System (DBMS).

Figure 5.1 illustrates the server design in a simplified way. The acceptor thread opens the server socket and waits for incoming connections. Requests from clients are then handed over to a free worker thread, chosen from a pool of available ones. Worker threads in turn fetch the command identifier from the request and execute a concrete *Command* implementation. Most of the commands communicate with the local `SQLite` database using Java Database Connectivity (JDBC). Once obtained, the requested information is then returned to the client. The *Commitment Updaters* are another set of threads who continuously update the trust commitments with other

[1] http://javafx.com
[2] http://sqlite.org

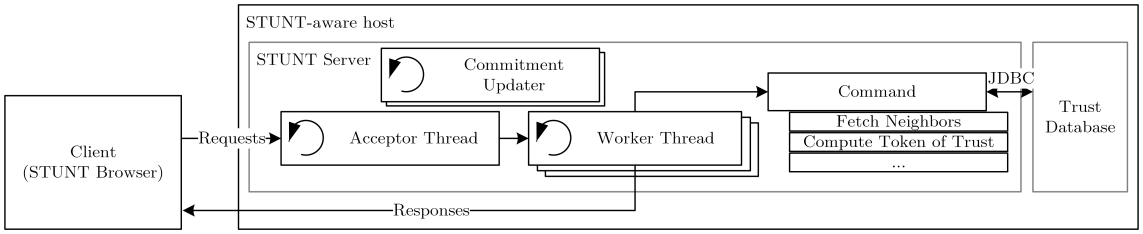


Figure 5.1: STUNT server architecture

hosts, in intervals defined during the trust relationship establishment. There is one thread per trust relationship, since the update intervals may differ from one relationship to another.

Control Interface The control interface is another interface to the database. A Graphical User Interface (GUI) and a command line interface are available. The latter one is useful when automating STUNT tasks.

Figure 5.2 shows a screenshot of the GUI. It is kept simple, with just an overview about both existing trust relationships and the audit trail. The existing trust relationships are indicated green or red, depending on whether or not the last trust commitment was received in time. It might not be the case if the other party revoked the trust relationship, or their server service is simply down and thus not sending updates. New trust relationships can be established using the `Add new Trust` menu item. The user is then prompted to specify the hostname and an update interval, and a request is sent to that host, denoting the start of the protocol as described in Section 4.3.6.

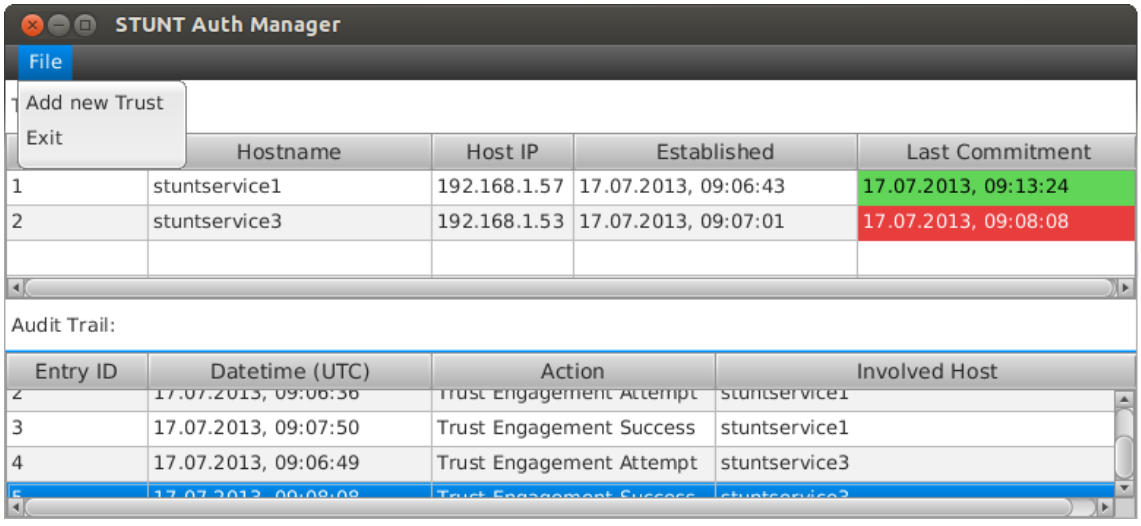


Figure 5.2: STUNT control interface GUI

The command line interface offers similar functionality, with the ability to add new trust relationships and view or revoke existing ones. Furthermore, one can view, accept or deny pending requests and inspect the audit trail from the command line.

The component design is simple, with two central threads who monitor the database for changes in the configuration of existing trust relationships and the audit trail to update the GUI in regular intervals. User commands are modeled using a command pattern, similar to the server component.

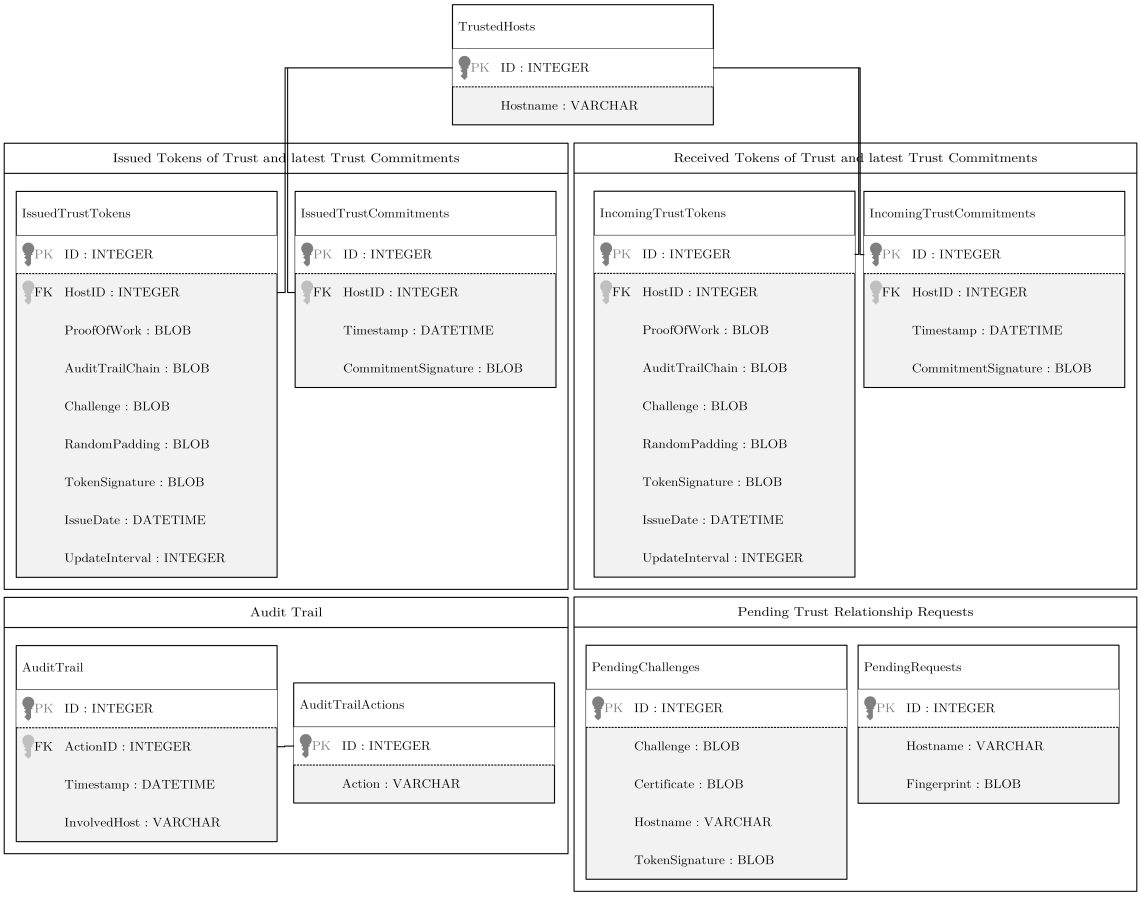


Figure 5.3: Server-side database model

User commands operate on the database, which acts as the interface between the control interface and STUNT server. Messages sent to other hosts are addressed to the other host's STUNT server.

Server Database The local trust database on server-side is implemented using `SQLite`, with a schema as depicted in Figure 5.3.

The model stores both the issued and received parts of the mutual trust relationships, all linked to a common table of *trusted hosts*. The tokens of trust, as well as the trust commitments, are a database representation of the structures defined in Section 4.3.6. Furthermore, the audit trail with its predefined possible entry types is stored. Two tables, called `PendingChallenges` and `PendingRequests`, temporarily store intermediate data when the establishment protocol is currently running.

Client Database The client database, referred to as personal *trust base*, is another `SQLite` database. Figure 5.4 shows the database schema, which consists of only two tables. `VisitedHosts` contains entries of hosts who were assessed by the user, including those indirectly included as vouchers. Hosts may either be labeled as trusted, or just be noted as *references* to other, trusted hosts. If the user is presented with a cluster of hosts and decides to trust one node out of that cluster, the other, referencing nodes have to be stored as well. Otherwise, the client cannot

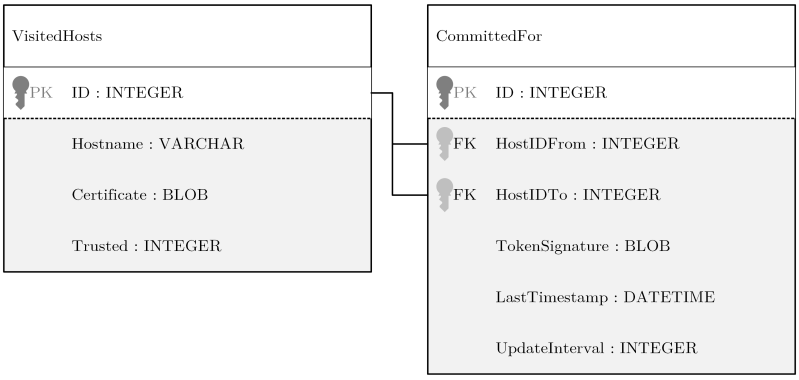


Figure 5.4: Client-side database model

determine whether the trust relationship configuration was changed on subsequent visits. Not storing this information would mean to not revalidate the trust network, which is undesirable.

Visited hosts may, but need not, have trust relationships with other hosts, which is modeled using the `CommittedFor` table. The `CommittedFor` table expresses the trust relationships on a database level, by linking two `VisitedHosts` and storing the token of trust signature, the last commitment timestamp and the corresponding update interval alongside this link. Note that a client might skip certain verification steps if the locally stored timestamp of the last commitment is within the bounds of the update interval, for example on frequent visits of the same host.

Client Browser The custom browser, written in Java with a `JavaFX` UI, is shown in Figure 5.5. As is depicted there, it is a rudimentary web browser with a simple user interface, supporting back/forward navigation and browser tabs. STUNT is embedded passively, using the `Explore` button on the top right. If the browser is pointed to a STUNT-aware host, it automatically performs the verification procedure as introduced in Section 4.3.7, and hints the user about the state of the visited host. Details about how the user is incorporated into the host authenticity verification are discussed in Section 5.3.

Similar to the other components, an asynchronous, event-driven command pattern is used to handle the user interactions. Note that for hypertext markup and stylesheet rendering, a `WebKit`-based[3], ready-made `JavaFX` component was used.

Component Interaction Communication with the corresponding databases is done only locally, using a common `JDBC` driver for `SQLite`, the *Xerial SQLite JDBC Driver*[4]. Network communication between the components is performed solely over SSL, using Java's built-in SSL support. The message formats are defined using Google's *Protocol Buffers*[5]. Protocol Buffers allows the specification of messages using its own specification language. From such message definitions, Java code is generated, which allows efficient serialization and de-serialization, while ensuring non-ambiguity and resource-saving encoding.

[3] http://webkit.org
[4] https://bitbucket.org/xerial/sqlite-jdbc
[5] https://developers.google.com/protocol-buffers/

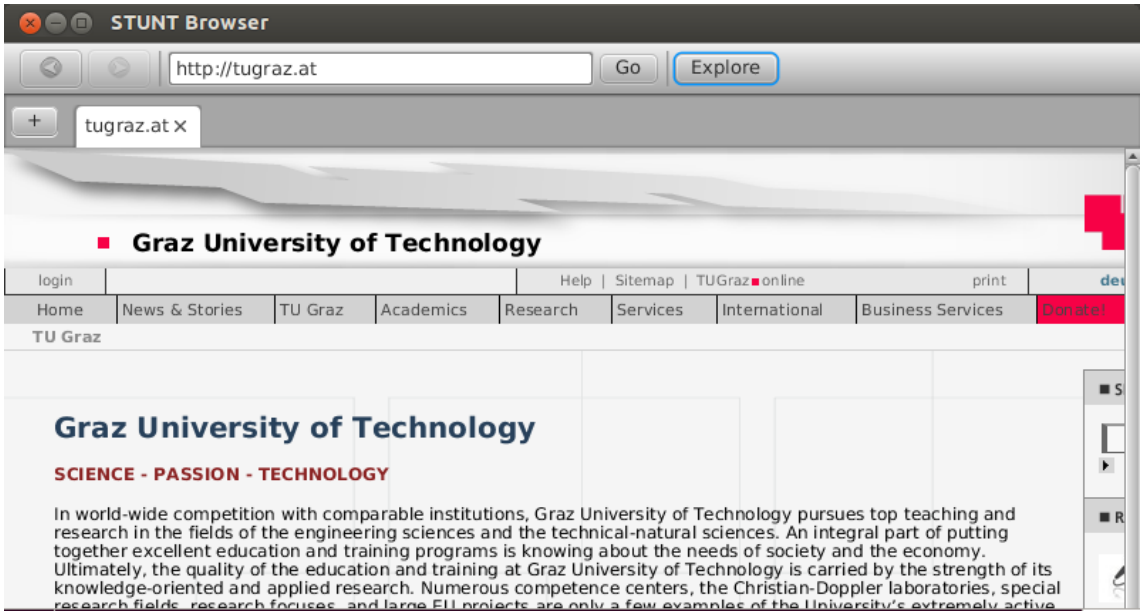


Figure 5.5: STUNT browser

5.2 Lab Setup and Evaluations

To test the system in a small-scale way, a set of virtual machines was set up and configured to form a STUNT trust network. The lab setup is certainly by no means as extensive as a real-world Internet context, but the dimension is assumed to be sufficient to get first impressions on how the system scales.

The lab setup consists of 13 virtual machines, connected as depicted in 5.6. Additionally, there is one server Virtual Machine (VM), responsible for Dynamic Host Configuration Protocol (DHCP)-based IP address assignment and DNS resolving, as well as one machine with the browser installed, acting as the client.

The host names loosely resemble renowned Austrian domains from the public and academic sector, such as certain web sites of austrian ministries and universities. All the virtual machines are hosted on one `VMWare ESXi`[6] server with only two 2 GHz CPUs. We consider STUNT to be performing acceptably fast, despite the low hardware dimensions. Verification times are indeed negligible, as the next paragraph shows, and fetching the neighborhood information from a node is fast as well. Concrete timings of the latter were not performed, since it highly depends on network latency, but with common networking delays being usually well below 100ms, according to AT&T [2013], delays are not considered a general problem.

To support the claim of cheap verification, the verification procedure for newly visited hosts is simulated on a Google Nexus One mobile phone with a 1 GHz CPU and 512 MB of memory, using Android[7] 2.3 as operating system. The verification of the immediate trust network of a host having 10 trust relationships and an audit trail length of 100 entries takes 132.85 ms, averaged over 100 measurements. We intentionally used an outdated, constrained device, and we expect

[6] http://www.vmware.com/products/vsphere-hypervisor/overview.html
[7] http://android.com

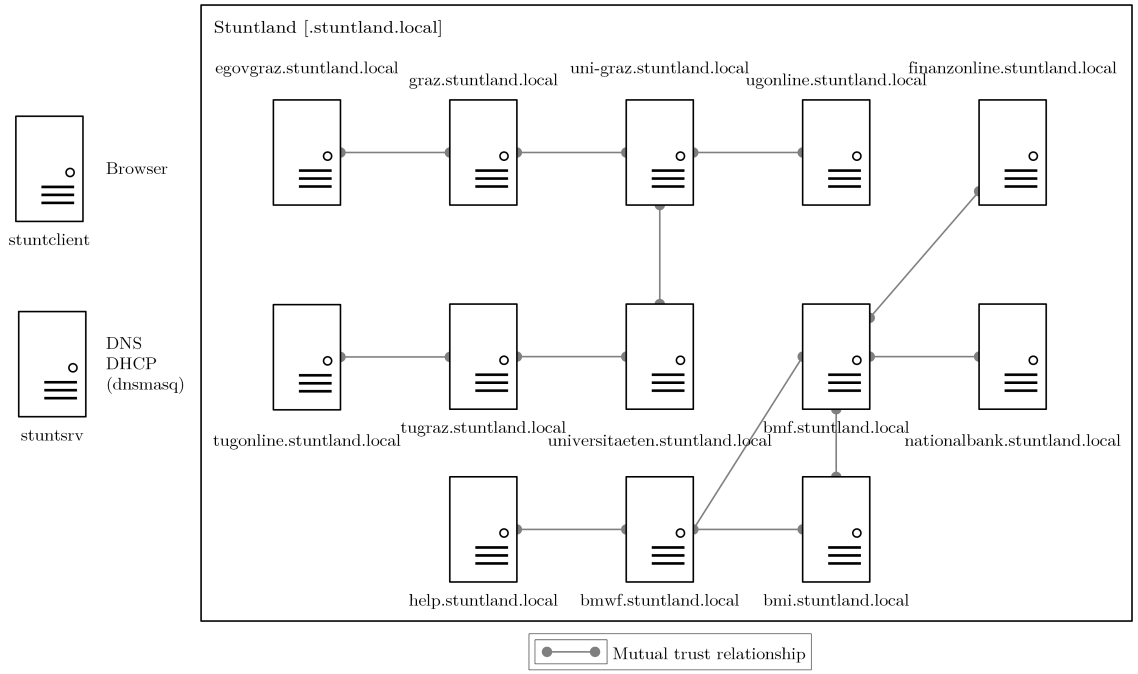


Figure 5.6: STUNT lab setup

desktop verification times to be way below that value. Furthermore, we evaluated this scenario with a high number of incoming trust relationships (10), by configuring STUNT with a lower difficulty for establishing trust relationships. Note that lowering the proof of work difficulty does not lower the verification complexity. We expect the regular case to have only 5–6 incoming trust relationships, and this is an attempt to further exacerbate the complexity of the problem. Therefore, the verification time almost solely depends on network delays, with computation time being negligible.

We consider the storage space required on client-side to save local copies of subsets of trust networks low as well. The required storage space for 50 hosts, with each host having 10 distinct trust relationships and all involved hosts using RSA key lengths of 4096 bit, is extrapolated from data gathered from the lab setup. With these 50 hosts, which we consider a high estimate for a regular user, the storage space required is less than 2.3 megabytes.

5.3 User Integration

The success of STUNT is dependent on the quality of the user interface. More precisely, it is highly depending on user acceptance in general. Designing an intuitive, appealing user interface is considered future work, which requires user interface expertise. This thesis focuses on the core mechanisms of the system itself. Nevertheless, a rudimentary user interface for network exploration is implemented, to get a feeling about whether or not STUNT is a viable alternative.

Figure 5.7 shows the proof-of-concept realization within the browser component. On the left hand side is a sample web content. On the right hand side is the STUNT-related information associated with the visited host, `graz.stuntland.local`. The panel on the right is toggled with the `Explore`

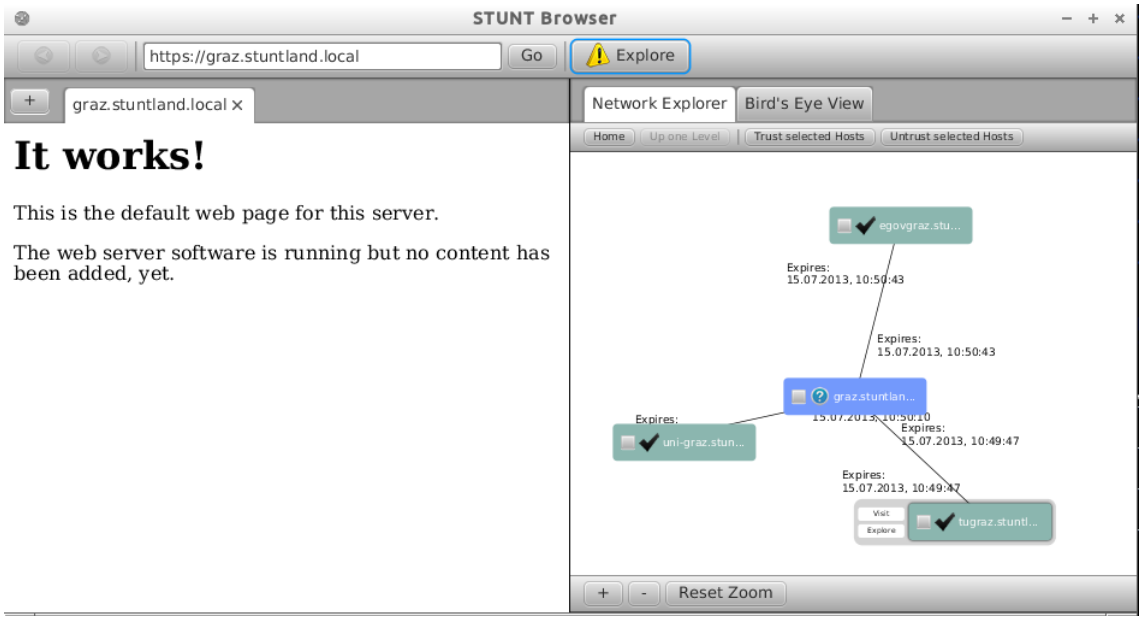


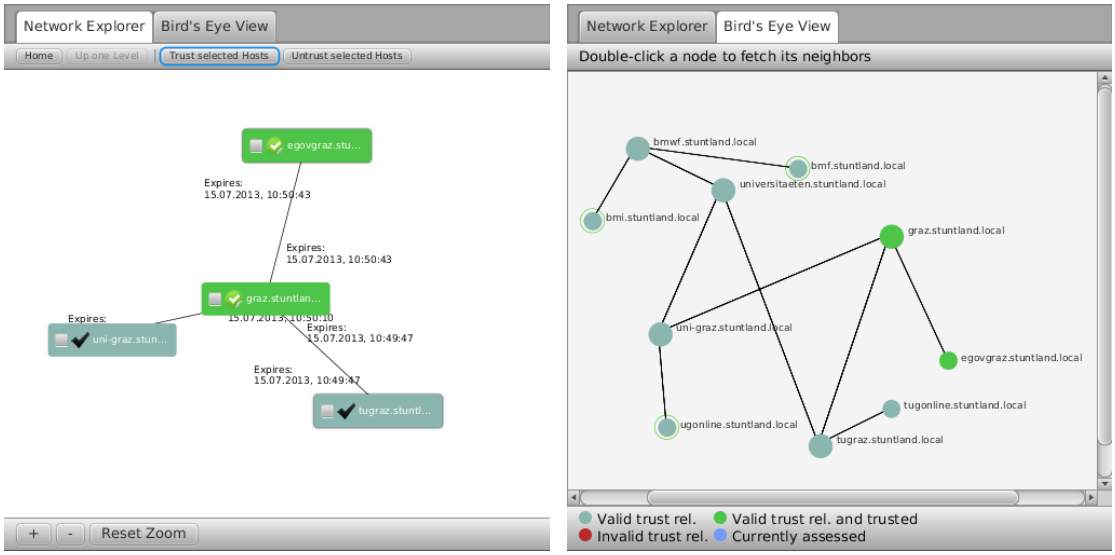
Figure 5.7: The integrated network explorer, level-based view

button on the top right of the browser window. The button is currently showing a warning sign, which indicates that the host's trustworthiness was never acknowledged by the user. Therefore, the prototypical browser uses a passive-only approach, where the user is not interrupted by means of a warning, such as is the case in current SSL browser notifications. It is certainly less annoying, but bears the danger of not being recognized at all.

The content of the panel to the right shows the hosts directly trusting the visited host. The visited host is centered, and the relationship connectors indicate when the trust relationships expire. The surrounding nodes may be rendered in different colors and with different items, depending on the state of the trust relationship. In the figure, all trust relationships verified correctly, which is why all nodes are rendered turquoise and show a checkmark. If a verification fails, the corresponding node is colored in red and marked with an X. Nodes who are already within the personal trust base appear in green color and show an encircled checkmark.

Figure 5.8 shows the STUNT button states. The button is green and shows a checkmark if the host was already added to the personal trust base, and the trust relationships are still in the same state as they were upon the initial assessment. The button is red and shows an X if a trust relationship verification has failed, or if one or more trust relationships are missing, compared to the state when the host was initially assessed. A missing trust relationship means that the target host was unable to deliver a current trust commitment, which might be due to a revocation of this trust relationship. A warning icon is displayed if either the host was never visited before, or if the current state differs to the original state in a non-critical way. For example, there might be a new trust relationship, which is not considered a bad thing. Nevertheless, the user is still hinted to

Figure 5.8: Graphical button representations of the STUNT states



(a) The integrated network explorer, level-based view with trusted hosts

(b) Bird's eye view of the same model

Figure 5.9: Network explorer and bird's eye view in comparison

inform about this change of state.

The view in Figure 5.7 is multi-leveled. A level describes one hierarchy stage in the tree-like representation of the network explorer. Initially, the first level is displayed, showing only directly connected nodes. Deeper exploration of the network requires stepping down the hierarchy, thus showing deeper levels. The user has the option to browse the network more deeply by hovering over a neighboring node, as shown in the lower right node in the figure. The option *visit* allows users to view the host's web site, whereas *explore* puts the hovered node to the center and displays the trusting hosts of the newly centered node. That way, users can browse the network arbitrarily deep. The controls *Home* and *Up one Level* in the upper toolbar allow navigating between the levels.

The buttons *Trust selected hosts* and *Untrust selected Hosts* handle the addition and removal of hosts to and from the personal trust base. Nodes whose checkbox has been selected are added or removed. Users can browse multiple levels, select those hosts whom they decide to label as trustworthy and add them all to the personal trust base in one go. Once labeled as trusted, they will appear green and show an encircled checkmark in the explorer, as is illustrated in Figure 5.9a.

The second view, referred to as *Bird's Eye View*, is depicted in Figure 5.9b. We intend the bird's eye view to be better for an overall view of the network, at the cost of omitting some details. However, it is a view on the same data model, and users can freely switch between the two. The bird's eye view is a reduced view, designed to display only immediately relevant data, whilst giving a larger-scale overview of the system. The zoomed-out perspective of the bird's eye view allows users to detect suspicious, very limited trust networks, with just a couple of connected hosts. Initially, only two levels of trust relationships are shown to ensure the presented information is relevant. On demand, users can then double-click any remote nodes to have the network further expanded in a dynamic way. Nodes whose neighbors are not yet fully fetched are indicated with

a green circle. Therefore, the user can interactively browse the network and expand it to point of interests. Furthermore, delays caused by the network are not immediately noticeable, due to the low number of nodes fetched in parallel.

The nodes change their size with the number of other connected, trusting nodes. A bigger node means it is better embedded in the trust network than a smaller node. The shown graph is drawn using a force-directed layout algorithm as introduced by Fruchterman and Reingold [1991] to minimize edge overlap and cluster nodes reasonably. Users can, however, additionally move nodes by themselves using drag and drop. Furthermore, both views are zoomable to allow seamless navigation.

The user interface is prototypical, with room for improvement. It would certainly be of interest if such a passive approach is sufficient, or active measurements, like interrupting warnings, are the way to go. If the user interface is appealing, it might even be *interesting* for users to browse the trust network. If a host is important enough to be validated this way, it might well be exciting to know whom its operators partnered with.

5.3.1 Device Synchronization

Nowadays, with smartphones, tablets and other gadgets being part of our everyday lives, it is likely that users want to synchronize their local trust base(s) to be consistent among multiple devices. From a user experience point of view, this is desirable, since it counteracts boring re-assessments of the same service. Synchronization between multiple devices is considered possible, since the local trust bases are merely plain `SQLite` databases. Firefox, for example, uses multiple local `SQLite` databases, and comes with a service to synchronize user-data among devices, called *Firefox-Sync*[8]. Currently, it supports synchronizing open tabs, bookmarks, passwords, the browsing history, add-ons and preferences, some of which are already stored in `SQLite` databases. The synchronization service uses a shared, secret token to tie the individual devices to one account. Given that the very same thing is already done in Firefox and other browsers, it is considered manageable to synchronize personal trust bases between multiple devices in a feasible, protected way.

[8] See https://support.mozilla.org/en-US/kb/firefox-sync-take-your-bookmarks-and-tabs-with-you

Chapter 6
Discussion

"But we had forgotten that alongside Orwell's dark vision, there was another — slightly older, slightly less well known, equally chilling: Aldous Huxley's Brave New World. Contrary to common belief even among the educated, Huxley and Orwell did not prophesy the same thing. Orwell warns that we will be overcome by an externally imposed oppression. But in Huxley's vision, no Big Brother is required to deprive people of their autonomy, maturity and history. As he saw it, people will come to love their oppression, to adore the technologies that undo their capacities to think."

– Neil Postman

STUNT is an incipient alternative to the currently established PKIX infrastructure for verifying host authenticity in a large network context. The crux of STUNT is, intentionally, the user, and the system will only work if users are able and willing to actually use it. Users have to be successfully encouraged to use this verification procedure for their own benefit, which might be possible, if the mechanism is appealing enough. If designed in an appealing way, it might even be exciting to browse trust relationships and thus learn more about the counterpart. If this can be accomplished, users may largely benefit from the dynamic, individual trust bases that emerge. We consider individual trust bases, which may easily be synchronized between one's devices, an improvement over predefined, intransparent lists of automatically trusted roots. Involving users in the decision-making in the first place will additionally ease the side-effect that users do not know how to react to warnings, because they assessed a host's trustworthiness in the first place. Therefore, they already have an initial idea about the problem, and can take according measures. Users cannot be sure that the presented network of trust is indeed genuine, but deep exploration alongside plausibility checks regarding established trust relationships render the chance of a fake network very low.

To the best of our knowledge, the proposed configuration of cryptographic primitives to achieve the introduced properties of *user-centrality*, *mutual trust relationships*, *expensive trust relationship establishment*, *cheap trust relationship verification*, *risk-based revocation support* and what we called the *backfiring principle* is new and enables a sparse, meaningful network of trust. In contrast to PGP's web of trust, we put more focus on the value of individual trust relationships, and we took measures so they cannot arbitrarily be given away without harming oneself. This leads to meaningful trust expressions and a low, graspable number of references to verify. In comparison to PKIX, it bears the advantage of autonomous trust decisions, without depending on unknown, but *trusted* third parties. Moreover, it is considered to be more robust, since the impact of a misbehaving host is limited, due to the ability to quickly cancel trust relationships. In contrast, a misbehaving CA is hardly replaceable or revokeable, as it would affect all hosts whom this CA vouches for. The creation of very powerful, single nodes is intentionally prevented to avoid the

concentration of power and thus limit the possible impact in case such a node fails.

Nevertheless, well-known nodes, such as Google for example, might attract numerous operators and thus have the need to establish more trust relationships than what is allowed by the system. To circumvent this limitation, it is possible to create a semi-internal network of trust. For example, `google.com` might trust `mail.google.com` and `maps.google.com`, which in turn all trust external partners. Such a scenario leads to a similar, hierarchic structure as in PKIX, but with two key differences:

1. The system does not fail if, hypothetically, `Google`, the CA-equivalent in the example, is no longer trusted. In a PKIX world, removing a top-level CA from the list of trusted roots would render all the certificates it issued untrusted. Therefore, it would result in intransparent warnings at the user's. These warnings interrupt an otherwise automatic process, which users usually don't see working at all. In STUNT, users are warned as well, but about an anomaly of something *they assessed themselves*, thus resulting in better awareness of the problem.

2. All operators are *free to choose* with whom they establish a trust relationship. It means that, instead of being required to choose an accredited TTP, operators can freely select partners whom *they* trust. In case of doubt, established trust relationships can be revoked within a pre-defined risk interval.

Thus, there may be structures similar to the hierarchic approach of PKIX within the STUNT network. These are, however, *chosen* to be present. If operators decide to have it that way, and users accredit this, then so be it. If users reject such constructs, and categorize a host as untrustworthy if it does not trust `google.com`, but instead, say, `www0127.google.com`, then such patterns are unlikely to emerge at all. It is likely that the operators of attractive hosts will charge money for establishing a trust relationship, or have them sold to the highest bidder during auctions. Less-known hosts may have to offer money to be trusted, as is for example done in sponsorship contracts. The payment will ensure that their host is shown to users everytime they evaluate the trust network. Such payments are neither part of the initial STUNT design, nor enforced in any way, as they blur the intended meaning of trust relationships. We consider such scenarios still better than in the current CA situation, because, as opposed to CAs, the selling of trust expressions is not the sole, core business of the operators. They are less dependent on selling trust relationships, and risk their own reputation by doing so. Therefore, we still think that careful choices are made when selecting potential partners. Besides, potential buyers still have the freedom to choose, and might as well opt for other host operators, who do not charge money for establishing a trust relationship. However, it is hard to predict future entrepreneurs' ideas from today's point of view, so there may be unrecognized problems attached to charging for a trust relationship. The general idea is to choose partners based on their *evaluation trust*, and both the mutuality and expensiveness support this type of evaluation.

Another considerable aspect regarding the proof of work computation is the difference of computation power of the involved machines. Having superior computation power certainly leads to an advantage in terms of the time a trust relationship establishment takes. The proof of work contains both endpoints' certificate fingerprints, and the establishment protocol already, intentionally, requires human interaction. Therefore, it is a small leap to externalize the proof of work

computation, and have it run on another machine, or a cloud service. The cloud provider cannot use the proof for its own purposes, since the certificate fingerprints will not match, and operators can resort to on-demand computing power to solve this intentionally hard task. All host operators have similar access to computational resources, by buying their own hardware, solving the problem using a cloud service, or simply accepting longer runtimes. The feature of easy externalization would ease the problem of establishing trust relationships for constrained hardware services as well. Furthermore, token versioning may be incorporated, to deal with the general increase of computation power over time. Token versioning enables setting a certain date, after which all issued tokens have to be of a specific version. A token version may be bound to a defined level of initial difficulty, and a rule to increase the difficulty, depending on the other party's amount of trust relationships. Thus, it provides a simple way to adapt the required effort to current system speeds and, as a result, ensure the trust relationships' expensiveness.

We consider the discussed attacks, *replacing* and *remodeling*, low, but non-negligible threats. Remodeling requires special attention from users during the host assessment to detect fraudulent copies of a trust network, which is supported by the *Bird's Eye View* GUI component in the prototype. Furthermore, it requires high computational power and patience from the adversary, works only on the first visit, and is considered likely to be detected soon, rendering its cost-benefit ratio poor. Replacing on the other hand requires host operators to pay attention to incoming trust relationship requests. Since their credibility is dependent on their trust relationships, host operators are assumed to make careful decisions. However, in case a trust relationship was established by accident, it can easily be revoked within the specified risk interval. Also, both attack vectors require extensive network control and thus very powerful attackers. Nevertheless, we do not know yet how users are actually interacting with the system. Therefore, we may underestimate the threats of these attacks from the current point of view.

The system is considered to work best when the emerging network of trust is connected as a whole, since remodeled clusters are detected easier when the network of trust is large. Multiple clusters should be sufficient as well, as long as their size is big enough to render a complete remodeling unmanageable. Having large components emerge is considered likely, even though the artificially constrained network of trust cannot have hubs, who connect numerous other nodes due to their high number of links. Contrary to typical, scale-free social networks, with a power-law degree distribution [Newman, 2003], the edges per node are limited to a constant in STUNT. The network is still assumed to have high connectedness, because of the incentive to actually *have* trust relationships. The number of edges is severly limited, but the individual nodes are encouraged to *have* edges to appear credible. It is the *nodes* that become rare. There is tension between availability and quality, leading to two differing results: a) host operators may opt to enter unreasonable trust relationships, instead of having none at all, and b) as a result, new host operators are more likely to get trust relationships established, despite the limitation. Host operators who carelessly establish trust relationships weaken the system as a whole. We think that having no trust relationships is certainly suspicious, but being trusted by entirely unrelated or otherwise strange partners is no less ominous. Moreover, we do not expect such trust relationships to last for long, even if established for these reasons, since they might be dropped as soon as there is a better option available. On the other hand, new players are likely to find nodes to attach to, since potential partners may just not have used all their available trust relationship slots yet. As an additional incentive to attach to new, less-prominent nodes, the system bases the computational

difficulty of the proof of work on the other party's amount of trust relationships. Therefore, it is computationally cheaper to engage in trust relationships with less-prominent hosts. How severe the problem of finding adequate links might become, however, cannot be answered for now.

The built-in revocation mechanism is active, which means it requires continuous effort to work. It has the advantage of a whitelisting-based verification, contrary to blacklisting of individual certificates. Nevertheless, it might lead to problems if the corresponding risk interval is chosen to be very low. Systems might have outages longer than the risk interval, which leads to omitted trust commitment updates, and thus to a seemingly revoked host. Once the previously unresponsive host is back up and sends a commitment, the trust relationship is valid again. In the meantime, however, users may experience confusing warnings. Moreover, the fact that multiple audit trails are unavoidable may lead to many categories, which might render the surrounding trusted hosts hardly graspable for users. We think that this problem may, to some extent, be overcome by smart user interface design, but after a certain number, it will indeed become a problem. This problem can only be overcome if user's are taught to be suspicious when hosts have a large number of trust communities.

STUNT is hard to integrate into the current network landscape. It certainly takes effort, since STUNT, even when operated in compatibility mode, requires a new certificate containing the STUNT public key, and an additional server service to be run on the target host. Client-side, the verification procedure has to be built into the browser software. A challenging part is to integrate an appealing user interface into current web browsers. In addition, STUNT certainly needs transition-times, where users get comfortable with the new system. We believe that the concept of evaluating trust relationships is considered simple enough to be usable without special training. However, it surely will take time for users to accomodate to evaluating trust relationships. As a benefit, we assume this simple mechanism to considerably raise the awareness about the authenticity problem.

We do think that we provided a new, fresh point of view to long-known problem of host authenticity. We also consider the system to be self-regulating if minor mistakes regarding the establishment of trust relationships occur. The mutuality of trust relationships should be incentive enough for host operators to revoke accidental trust relationships with unknown other parties. Revocation is an integral part of the system, with a whitelisting verification procedure, where no host is considered valid until all requirements are met. The power of a single node is delimited by the maximum amount of trust relationships, thus, the impact of one misbehaving node is limited as well. The system focuses on the user, and tries to provide a way of assessing hosts based on their trust relationships. Whether we are right in our positive assumptions about user behaviour can only be answered by thorough user experience analysis. Therefore, we cannot claim that this system will work as-is. We do think, however, that it is a step in the right direction, and hope that it stimulates new thoughts about mitigating the authenticity problem.

Chapter 7

Conclusion and Future Work

"If you want to truly understand something, try to change it."
– Kurt Lewin

STUNT is an approach towards ensuring host authenticity in an Internet context. As such, it allows users to decide upon the authenticity of a host by means of assessing the expressed trust of other hosts. Therefore, it uses a comparable concept to checking reference partners when contracting with a previously unknown company.

The system cryptographically ensures that a reference to another, legitimate host cannot be spoofed, and incorporates means to have bad behaviour backfire at the initiator. Furthermore, host operators are encouraged to collect as many trust relationships as possible, with the maximum number of trust relationships being severely limited at the same time. Host operators are intentionally put into this catch-22 situation to have them think twice about whom they engage in a trust relationship with. Trust relationships are expensive to establish to encourage continuous agreements and thus not have users interrupted frequently. On the other hand, trust relationships can be cancelled within a self-defined, reasonable risk interval. This risk interval is negotiated during the establishment of the trust relationship, where both parties have to agree upon a value. The interval defines the maximum possible time difference until users recognize a revocation and may be as low as a couple of minutes. STUNT is a decentralized system, which prevents single nodes from becoming substantially more powerful than others, and thus becoming a liability if their integrity is compromised.

Developing a concrete STUNT prototype led to several interesting insights, discussed throughout this work, and certainly to even more questions. Questions that might only be answerable via large, practical experiments. The most pressing ones of which are, in our opinion, questions regarding the following aspects:

User Acceptance STUNT succeeds or fails, depending on the user's willingness and capability to assess authenticity by means of attached trust relationships. If users reject this method, which is more work than in the current approach, then the approach in its current form might be rendered void.

User Interface Design Closely related to user acceptance is the question about how a user interface would have to look like to be accepted by users. We have shown a simple prototype implementation, but a real-world implementation, embedded into current browsers, would need more detail work.

Backfiring Principle We do not yet know if the mechanism we used to achieve the property called *backfiring principle* does indeed lead to the intended results. It might be

that users simply do not care whom a node is connected to, as long as there are connections. If that is the case, then choosing trusting nodes carelessly has no negative effects for the initiator, which in turn might reduce the over quality of trust relationships.

Parametrization The optimal difficulty parameterization of the STUNT proof of work system, considering vastly differing systems, is not to be answered trivially, and certainly involves a trade-off between the amount of supported devices and the targeted time spans.

Trust Communities The fact that hosts can maintain several audit trails in parallel, called *trust communities*, may become a problem if severely used. A multitude of categorizations, each incorporating trusting hosts, may overwhelm users and make them give up on assessing a host.

Therefore, we cannot claim that STUNT is a perfect, all-covering solution, although we still consider it worth pursuing. We believe that, given a tool as simple as the prototypical client implementation, people are capable of assessing host authenticity. This assessment is certainly more work for users than there is in the fully automatic approach of PKIX. The benefit is, however, that people would become aware of the problem, and the fact that there is no absolute guarantee for host authenticity. This is especially important in scenarios where automatic assurance fails. We consider it more likely to have users react upon STUNT warnings in an informed way, since they assessed the host's trustworthiness beforehand, and thus have an idea about what can go wrong. In PKIX, people are urged to respond to a warning about a process they were not involved in until it failed, leading to confusion and even ignorance.

We think that the next step should certainly be a set of user acceptance tests. Users have to be both willing and capable to use the system as intended, which is not yet proven. Large-scale user acceptance tests should also reveal if the backfiring principle is a valid assumption. Moreover, we believe that substantial work has to be put into designing an appealing user interface, which does not repel users in the first place. The parametrization may be overcome with token versioning, but the optimal choice of difficulty per interval still remains an interesting aspect of future work.

We did not only choose the acronym STUNT, a Simple, Transparent, User-centered Network of Trust, because its individual letters resemble the cornerstones of the system, but also because it is indeed a *stunt* to challenge a system as deeply entangled into the Internet as PKIX. Nevertheless, we consider the approach viable and having users informed about what happens within a browser behind the screen a desirable goal. We intentionally use a concept as simple as inspecting a very limited number of unqualified trust relationships to not overwhelm users and keep the overhead in workload at a minimum. We think STUNT is a concept capable of providing a fresh view and a possible, though bumpy path to a better mitigation mechanism than currently adopted approaches.

STUNT is an effort to include the user in security-relevant decisions. Especially the decision about another's trustworthiness is not a delegatable one. Therefore, the benefits for users are assumed to outweigh what we expect to be a small number of additional interactions. The system is designed in a way to minimize user assertions, with the ability to store user's decisions in a local trust database, which is personal per user and can be synchronized among multiple devices. Therefore, we think it is a convenient approach, with little overhead for users, whilst still incorporating measures for

reliable authenticity assessment.

Bibliography

Abdessalem, T., B. Cautis, and A. Souhli (2010, January). Trust Management in Social Networks. http://isicil.inria.fr/v2/res/docs/livrables/ISICIL-ANR-EA05-TrustManagement-1001.pdf. Information Semantic Integration through Communities of Intelligence onLine (ISICIL) State of the Art Publication.

Abdul-Rahman, A. and S. Hailes (1997). A Distributed Trust Model. In *Proceedings of the 1997 Workshop on New Security Paradigms*, NSPW '97, New York, NY, USA, pp. 48–60. ACM.

Adams, A. and M. A. Sasse (1999, December). Users are not the Enemy. *Commun. ACM 42*(12), 40–46.

Adams, C. and S. Lloyd (2002). *Understanding PKI: Concepts, Standards, and Deployment Considerations* (2 ed.). Addison-Wesley Professional.

Albertson, T. (2013, June). Windows Root Certificate Program - Members List (All CAs). http://social.technet.microsoft.com/wiki/contents/articles/14215.windows-and-windows-phone-8-ssl-root-certificate-program-member-cas.aspx.

Amann, B., M. Vallentin, S. Hall, and R. Sommer (2012, November). Extracting Certificates from Live Traffic: A Near Real Time SSL Notary Service. Technical report. International Computer Science Institute, Berkeley.

Amann, B., M. Vallentin, S. Hall, and R. Sommer (2013, June). The ICSI Certificate Notary. http://notary.icsi.berkeley.edu/. International Computer Science Institute, Berkeley.

Arends, R., R. Austein, M. Larson, D. Massey, and S. Rose (2005a, March). DNS Security Introduction and Requirements. https://www.ietf.org/rfc/rfc4033.txt. Internet Engineering Task Force.

Arends, R., R. Austein, M. Larson, D. Massey, and S. Rose (2005b, March). Protocol Modifications for the DNS Security Extensions. https://www.ietf.org/rfc/rfc4035.txt. Internet Engineering Task Force.

Arends, R., R. Austein, M. Larson, D. Massey, and S. Rose (2005c, March). Resource Records for the DNS Security Extensions. https://www.ietf.org/rfc/rfc4034.txt. Internet Engineering Task Force.

Arnbak, A. and N. Van Eijk (2012, August). Certificate Authority Collapse: Regulating Systemic Vulnerabilities in the HTTPS Value Chain. http://dx.doi.org/10.2139/ssrn.2031409. Available at SSRN.

Artz, D. and Y. Gil (2007, June). A survey of trust in computer science and the semantic web. *Web Sem. 5*(2), 58–71.

Asghari, H., M. Van Eeten, A. Arnbak, and N. Van Eijk (2013, June). Security Economics in the HTTPS Value Chain. http://dx.doi.org/10.2139/ssrn.2277806. Available at SSRN.

Ashri, R., S. D. Ramchurn, J. Sabater, M. Luck, and N. R. Jennings (2005). Trust Evaluation through Relationship Analysis. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, New York, NY, USA, pp. 1005–1011. ACM.

AT&T (2013). Global Network Latency Averages. http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html.

Back, A. (2002). Hashcash - A Denial of Service Counter-Measure. http://www.hashcash.org/papers/hashcash.pdf. Technical Report.

Bamberger, W. (2010). Interpersonal Trust – Attempt of a Definition. http://www.ldv.ei.tum.de/en/research/fidens/interpersonal-trust/. München University of Technology.

Bartsch, S., M. Volkamer, H. Theuerling, and F. Karayumak (2013). Contextualized Web Warnings, and How They Cause Distrust. In M. Huth, N. Asokan, S. Čapkun, I. Flechais, and L. Coles-Kemp (Eds.), *Trust and Trustworthy Computing*, Volume 7904 of *Lecture Notes in Computer Science*, pp. 205–222. Springer, Berlin Heidelberg.

Baumgartner, K. (2012, February). When Certificate Authority Business Models and Vendor Certificate Policies Clash. https://www.securelist.com/en/blog/208193386/When_Certificate_Authority_Business_Models_and_Vendor_Certificate_Policies_Clash. Kaspersky Labs.

Binst, J. (2011, September). VASCO Announces Bankruptcy Filing by DigiNotar B.V. http://www.vasco.com/company/about_vasco/press_room/news_archive/2011/news_vasco_announces_bankruptcy_filing_by_diginotar_bv.aspx. VASCO.

Blaze, M., J. Feigenbaum, and A. D. Keromytis (1999). KeyNote: Trust Management for Public-Key Infrastructures. In *Proceedings of the 6th International Workshop on Security Protocols*, London, UK, UK, pp. 59–63. Springer-Verlag.

Blaze, M., J. Feigenbaum, and J. Lacy (1996). Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Washington DC, USA, pp. 164–173. IEEE Computer Society.

Boeyen, S. and Moses, T. (2003, January). Trust Management in the Public-Key Infrastructure. http://download.entrust.com/resources/download.cfm/21126/. Entrust.

Brands, S. A. (2000). *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press.

Brown, D. R. L. (2009). Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography. Technical report, Certicom Corp.

Carminati, B., E. Ferrari, and A. Perego (2009, November). Enforcing Access Control in Web-based Social Networks. *ACM Trans. Inf. Syst. Secur. 13*(1), 6:1–6:38.

Chu, Y.-H., J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss (1997). REFEREE: Trust Management for Web Applications. *World Wide Web J. 2*(3), 127–139.

Clarke, D., J. E. Elien, M. Fredette, A. Morcos, and R. Rivest (2001). Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security 9*(4), 285–322.

Clarke, R. (2001). The fundamental inadequacies of conventional public key infrastructure. In *Proceedings of the 9th European Conference on Information Systems, Global Co-operation in the New Millennium, ECIS 2001, Bled, Slovenia*, pp. 148–159.

Comodo (2011, March). Comodo Report of Incident - Comodo detected and thwarted an intrusion on 26-MAR-2011. http://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html.

Cooper, D., S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk (2008, May). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://www.ietf.org/rfc/rfc5280.txt. Internet Engineering Task Force.

Dang, Q., R. M. Blank, and P. Gallagher (2012). Recommendation for Applications Using Approved Hash Algorithms, NIST Special Publication 800-107, Rev. 1. Technical report.

Das, T. K. and B.-S. Teng (2004). The Risk-Based View of Trust: A Conceptual Framework. *Journal of Business and Psychology 19*(1), 85–116.

Datta, A., M. Hauswirth, and K. Aberer (2003). Beyond "web of trust": Enabling P2P E-commerce. In *Proceedings of the IEEE Conference on Electronic Commerce (CEC'03*, pp. 24–27.

Denning, D. E. (1993). A New Paradigm for Trusted Systems. In *Proceedings on the 1992-1993 workshop on New security paradigms*, NSPW '92-93, New York, NY, USA, pp. 36–41. ACM.

Dolev, D. and A. C. Yao (1981). On the security of public key protocols. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, pp. 350–357. IEEE Computer Society.

Dubois, T., J. Golbeck, and A. Srinivasan (2011, October). Predicting Trust and Distrust in Social Networks. In *Privacy, Security, Risk and Trust, 2011 IEEE Third International Conference on Social computing (SOCIALCOM)*, pp. 418 –424.

Eckersley, P. (2012, June). Sovereign Key Cryptography for Internet Domains, Design Document. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt. Electronic Frontier Foundation.

Eckersley, P., J. Burns, and C. Palmer (2010, August). The EFF SSL Observatory. https://www.eff.org/observatory. Electronic Frontier Foundation.

El Gamal, T. (1985). A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, New York, NY, USA, pp. 10–18. Springer.

Ellison, C. and B. Schneier (2000). Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal 16*(1), 1–7.

Ellison, C. M. (2011). SPKI. In *Encyclopedia of Cryptography and Security* (2 ed.)., pp. 1243–1245. Springer, Heidelberg.

Ellison, C. M., B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen (1999, September). SPKI Certificate Theory. https://www.ietf.org/rfc/rfc2693.txt. Internet Engineering Task Force.

European Commission, COM(2012) 238/2 (2012, June). Regulation of the European Parliament and of the Council: On Electronic Identification and Trusted Services for Electronic Transactions in the Internal Market, Proposal. http://europa.eu/rapid/press-release_IP-12-558_en.htm. Draft available at http://extranet.cor.europa.eu/subsidiarity/Lists/SmnItemsList/Attachments/3056/com_2012_2038_en.pdf.

Evans, C., C. Palmer, and R. Sleevi (2013, June). Public Key Pinning Extension for HTTP, IETF Draft, Version 6. https://tools.ietf.org/id/draft-ietf-websec-key-pinning-06.txt. Internet Engineering Task Force.

Felten, E. and G. McGraw (1999). *Securing Java*. John Wiley and Sons.

Froehlich, A. (2012, May). Fresh Model Needed for Broken SSL System. http://www.enterpriseefficiency.com/author.asp?section_id=1076&doc_id=237176. Dell Enterprise Efficiency.

Fruchterman, T. M. J. and E. M. Reingold (1991). Graph Drawing by Force-Directed Placement. *Software: Practice and Experience 21*(11), 1129–1164.

Furnell, S. and N. Clarke (2012). Power to the people? The Evolving Recognition of Human Aspects of Security. *Computers & Security 31*(8), 983–988.

Georgiev, M., S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov (2012). The most dangerous code in the world: validating ssl certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, pp. 38–49. ACM.

Golbeck, J. (2006, January). Trust on the World Wide Web: A Survey. *Foundations and Trends in Web Science 1*(2), 131–197.

Goodin, D. (2011a, April). How is SSL hopelessly broken? Let us count the ways. http://www.theregister.co.uk/2011/04/11/state_of_ssl_analysis/. The Register.

Goodin, D. (2011b, November). Web credential authority rebuked for 'poor' security. http://www.theregister.co.uk/2011/11/03/certificate_authority_banished/. The Register.

Grandison, T. and M. Sloman (2000, October). A Survey of Trust in Internet Applications. *IEEE Communications Surveys & Tutorials 3*(4), 2–16.

Gutmann, P. (2002). PKI: It's Not Dead, Just Resting. *Computer 35*(8), 41–49.

Gutmann, P. (2013). Engineering Security (draft version of April 2013). www.cs.auckland.ac.nz/~pgut001/pubs/book.pdf.

Hallam-Baker, P. (2013, Match). The recent RA compromise. http://blogs.comodo.com/it-security/data-security/the-recent-ra-compromise.

Hayes, J. M. (1998). The Problem with Multiple Roots in Web Browsers – Certificate Masquerading. In *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Los Alamitos, CA, USA, pp. 306–312. IEEE Computer Society.

Henning, E. (2012, February). Trustwave issued a Man-in-the-Middle Certificate. http://h-online.com/-1429982. Heise Online.

Herley, C. (2009). So Long, and No Thanks for the Externalities: The Rational Rejection of Security Advice by Users. In *Proceedings of the 2009 workshop on New security paradigms workshop*, NSPW '09, New York, NY, USA, pp. 133–144. ACM.

Hoffman, P. and J. Schlyter (2012, August). The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. https://www.ietf.org/rfc/rfc6698.txt. Internet Engineering Task Force.

Holz, R., L. Braun, N. Kammenhuber, and G. Carle (2011). The SSL Landscape: A Thorough Analysis of the X.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, New York, NY, USA, pp. 427–444. ACM.

Hoogstraaten, H., R. Prins, D. Niggebrugge, D. Heppener, F. Groenewegen, J. Wettinck, K. Strooy, P. Arends, P. Pols, R. Kouprie, S. Moorrees, X. van Pelt, and Y. Zheng Hu (2012, August). Black Tulip - Fox-IT Report of the Investigation into the DigiNotar Certificate Authority Breach. http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf.

Housley, R. and Ford, W. and Polk, W. and Solo, D. (1999, January). Internet X.509 Public Key Infrastructure Certificate and CRL Profile. https://www.ietf.org/rfc/rfc2459.txt. Internet Engineering Task Force.

IETF PKIX Working Group (2013, June). List of published PKIX documents. https://datatracker.ietf.org/wg/pkix/. Internet Engineering Task Force.

ISO/IEC 27000 (2009). Information technology – Security techniques – Information Security Management Systems – Overview and Vocabulary. Technical report, International Standards Organization.

ISO/IEC 9797 (2011). Information technology – Security techniques – Message Authentication Codes (MACs). Technical report, International Standards Organization.

ITU-T (2009, October). X.509 : Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks. ITU-T.

Jøsang, A. (1996). The Right Type of Trust for Distributed Systems. In *In: Proceedings of the 1996 New Security Paradigms Workshop (NSPW)*, ACM.

Jøsang, A. (2007, September). Trust and Reputation Systems. In G. R. Aldini A. (Ed.), *Foundations of Security Analysis and Design IV, FOSAD 2006/2007 Tutorial Lectures*. Springer LNCS 4677.

Jøsang, A. (2011). Trust Management in Online Communities. In V. Wittke and H. Hanekop (Eds.), *New forms of collaborative production and innovation on the Internet - interdisciplinary perspectives*, SOFI Göttingen. University Press Göttingen.

Jøsang, A. (2013). PKI Trust Models. In A. Elci and et al. (Eds.), *Theory and Practice of Cryptography Solutions for Secure Information Systems (CRYPSIS)*. IGI Global.

Jøsang, A., R. Ismail, and C. Boyd (2007). A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems 43*(2), 618–644.

Joseffson, S. (2006, March). Storing Certificates in the Domain Name System (DNS). https://www.ietf.org/rfc/rfc4398.txt. Internet Engineering Task Force.

Juels, A. and J. G. Brainard (1999). Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In S. Kent (Ed.), *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, pp. 151–165. The Internet Society.

Kaminsky, D., M. L. Patterson, and L. Sassaman (2010). PKI Layer Cake: New Collision Attacks Against the Global x.509 Infrastructure. In *Proceedings of the 14th international conference on Financial Cryptography and Data Security*, Berlin, Heidelberg, pp. 289–303. Springer-Verlag.

Karamanian, A., S. Tenneti, and F. Dessart (2011). *PKI Uncovered: Certificate-Based Security Solutions for Next-Generation Networks* (1 ed.). Macmillan Technical Publishing.

Keizer, G. (2011, March). Solo Iranian hacker takes credit for Comodo certificate attack. https://www.computerworld.com/s/article/9215245/Solo_Iranian_hacker_takes_credit_for_Comodo_certificate_attack. Computerworld, Inc.

Kirsh, D. (1990). When is Information Explicitly Represented? In *Information, Language and Cognition - The Vancouver Studies in Cognitive Science.*, pp. 340–365. UBC Press.

Laurie, B., A. Langley, and E. Kasper (2013, April). Certificate Transparency, IETF Draft, Version 12. https://tools.ietf.org/id/draft-laurie-pki-sunlight-12.txt. Internet Engineering Task Force.

Lenstra, A. K., J. P. Hughes, M. Augier, W. B. Joppe, T. Kleinjung, and C. Wachter (2012). Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064. http://eprint.iacr.org/.

Marlinspike, M. (2011a, November). Convergence - an Agile, Distributed and Secure Strategy for Replacing Certificate Authorities. http://convergence.io/details.html. Thoughtcrime Labs.

Marlinspike, M. (2011b, April). SSL and the Future of Authenticity. http://www.thoughtcrime.org/blog/ssl-and-the-future-of-authenticity/. Thoughtcrime Labs.

Marlinspike, M. and T. Perrin (2013, January). Trust Assertions for Certificate Keys, IETF Draft, Version 2. https://tools.ietf.org/id/draft-perrin-tls-tack-02.txt. Internet Engineering Task Force.

McKnight, D. H. and N. L. Chervany (1996). The Meanings of Trust. http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=F3FB52E4DF647092D7FA0F708B270482?doi=10.1.1.155.1213&rep=rep1&type=pdf. University of Minnesota MIS Research Center Working Paper series, WP 96-04.

McKnight, D. H. and N. L. Chervany (2001). Trust and Distrust Definitions: One Bite at a Time. In *Proceedings of the workshop on Deception, Fraud, and Trust in Agent Societies held during the Autonomous Agents Conference: Trust in Cyber-societies, Integrating the Human and Artificial Perspectives*, London, UK, UK, pp. 27–54. Springer-Verlag.

Menn, J. (2012, February). Key Internet operator VeriSign hit by hackers. http://www.reuters.com/article/2012/02/02/us-hacking-verisign-idUSTRE8110Z820120202. Reuters.

Merkle, R. C. (1988). A Digital Signature Based on a Conventional Encryption Function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, London, UK, UK, pp. 369–378. Springer-Verlag.

Miniwatts Marketing Group (2013). World Internet Usage Statistics. http://www.internetworldstats.com/stats.htm.

Moreau, T. (1998, March). Thirteen Reasons to Say 'No' to Public Key Cryptography. http://www.connotech.com/13REAS.HTM. Connotech Experts-conseils, Inc.

Morselli, R., B. Bhattacharjee, J. Katz, and M. A. Marsh (2006, 2006/03/02/). KeyChains: A Decentralized Public-Key Infrastructure. *Technical Reports from UMIACS, UMIACS-TR-2006-12*.

Mozilla (2012). Mozilla CA Certificate Store. https://www.mozilla.org/projects/security/certs/.

Myers, M., R. Ankney, A. Malpani, S. Galperin, and C. Adams (1999, June). X.509 Internet Public Key Infrastructure Online Certificate Status Protocol. https://www.ietf.org/rfc/rfc2560.txt. Internet Engineering Task Force.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf. The Bitcoin Project.

Netcraft, Ltd. (2012, January). SSL Survey. http://www.netcraft.com/internet-data-mining/ssl-survey/.

Newman, M. E. J. (2003). The Structure and Function of Complex Networks. *SIAM REVIEW 45*, 167–256.

Nigg, E. (2008). Untrusted Certificates. https://blog.startcom.org/?p=145#comment-87. Personal Blog of Eddy Nigg, Founder of StartCom.

Oprisan, A. (2013). Bitcoin Mining on Amazon EC2 - don't do it. http://andrei.oprisan.com/bitcoin-mining-on-amazon-ec2-dont-do-it/. Oprisan.com.

Parker, D. B. (1998). *Fighting Computer Crime: A New Framework for Protecting Information*. Wiley Computer Publishing.

Potzmader, K., J. Winter, and D. Hein (2013). Stunt - a simple, transparent, user-centered network of trust. In *Proceedings of the 10th European Workshop on Public Key Infrastructures, Services and Applications*, Berlin, Heidelberg. Springer-Verlag. To appear.

Rivest, R. and B. Lampson (1996, September). SDSI – A Simple Distributed Security Infrastructure. http://people.csail.mit.edu/rivest/sdsi10.html. MIT Computer Science and Artificial Intelligence Laboratory.

Rivest, R. L., A. Shamir, and L. Adleman (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM 21*(2), 120–126.

Roessler, T. and A. Saldhana (2010). Web Security Context: User Interface Guidelines. http://www.w3.org/TR/wsc-ui/. World Wide Web Consortium (W3C).

Roosa, S. B. and S. Schultze (2013). Trust Darknet: Control and Compromise in the Internet's Certificate Authority Model. *IEEE Internet Computing 17*(3), 18–25.

Sabater, J. M. (2003). *Trust and Reputation for Agent Societies*. Ph. D. thesis, Institut d'Investigació en Intelligència Artificial.

Sasse, M. A., K. Krol, and M. Moroz (2012, October). Don't work. Can't Work? Why it's Time to Rethink Security Warnings. In *Proceedings of the 7th International Conference on Risk and Security of Internet and Systems (CriSIS)*, Los Alamitos, CA, USA, pp. 1–8. IEEE Computer Society.

Smetters, D. K. and R. E. Grinter (2002). Moving from the design of usable security technologies to the design of useful secure applications. In *Proceedings of the 2002 Workshop on New Security Paradigms*, NSPW '02, New York, NY, USA, pp. 82–89. ACM.

Soghoian, C. and S. Stamm (2012). Certified Lies: Detecting and Defeating Government Interception Attacks Against SSL. In *Proceedings of the 15th international conference on Financial Cryptography and Data Security*, FC'11, Berlin, Heidelberg, pp. 250–259. Springer-Verlag.

Sotirakopoulos, A., K. Hawkey, and K. Beznosov (2011). On the Challenges in Usable Security Lab Studies: Lessons Learned from Replicating a Study on SSL Warnings. In *Proceedings of the Seventh Symposium on Usable Privacy and Security*, SOUPS '11, New York, NY, USA, pp. 1–18. ACM.

Sotirov, A. and M. Zusman (2009). Breaking the Myths of Extended Validation SSL Certificates. BlackHat Briefings, 2009 https://www.blackhat.com/presentations/bh-usa-09/ZUSMAN/BHUSA09-Zusman-AttackExtSSL-SLIDES.pdf.

Steinbrecher, S., F. Pingel, and A. Juschka (2011). Digital privacy. Chapter Reputation Management as an Extension of Future Identity Management, pp. 557–568. Berlin, Heidelberg: Springer-Verlag.

StJohns, M. (2007, September). Automated Updates of DNS Security (DNSSEC) Trust Anchors. https://www.ietf.org/rfc/rfc5011.txt. Internet Engineering Task Force.

Sunshine, J., S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor (2009). Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th conference on USENIX security symposium*, Berkeley, CA, USA, pp. 399–416. USENIX Association.

Tickbox.net (2006). DirectGov - Internet Thinking. http://webarchive.nationalarchives.gov.uk/20061211095958/http://www.cabinetoffice.gov.uk/e-government/docs/direct_gov/final_tickbox_report.pdf.

Trusted Computing Group (2011). TPM Main Specification Level 2 Version 1.2, Revision 116. http://www.trustedcomputinggroup.org/files/static_page_files/72C26AB5-1A4B-B294-D002BC0B8C062FF6/TPM%20Main-Part%201%20Design%20Principles_v1.2_rev116_01032011.pdf.

Tung, L. (2011, March). Comodo root canal call as more hacks confirmed. http://www.itnews.com.au/News/252881,comodo-root-canal-call-as-more-hacks-confirmed.aspx. Haymarket Media.

von Eitzen, C. (2011, September). CA DigiNotar bankrupt after SSL certificate debacle. http://h-online.com/-1346983. Heise Online.

von Eitzen, C. (2012, November). The story of a crime: notes on the DigiNotar break-in. http://h-online.com/-1742025. Heise Online.

Walker-Morgan, D. (2012, February). Break-ins at domain registrar VeriSign in 2010. http://h-online.com/-1427493. Heise Online.

Wendlandt, D., D. Anderson, and A. Perrig (2008, June). Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *USENIX 2008 Annual Technical Conference*, Berkeley, CA, USA, pp. 321–334. USENIX Association.

Whitten, A. and J. D. Tygar (1999). Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99, Berkeley, CA, USA, pp. 14–14. USENIX Association.

Wisniewski, C. (2013, Januar). Turkish Certificate Authority screwup leads to attempted Google impersonation. http://nakedsecurity.sophos.com/2013/01/04/turkish-certificate-authority-screwup-leads-to-attempted-google-impersonation/. Sophos, Ltd.

Yahalom, R., B. Klein, and B. T. (1993). Trust Relationships in Secure Systems - A Distributed Authentication Perspective. In *Proceedings, IEEE Symposium on Research in Security and Privacy*, pp. 150–164.

Zetter, K. (2012, September). Hackers Breached Adobe Server in Order to Sign Their Malware. http://www.wired.com/threatlevel/2012/09/adobe-digital-cert-hacked/. Wired.com.

Zimmermann, P. (1995). *PGP: Source Code and Internals*. MIT Press.

Zimmermann, P. (2000). *An Introduction to Cryptography*. Netwok Associates, Inc.