DISSERTATION

Bootstrapping Cryptography on the Internet

submitted in fulfillment of the requirements for the degree of
Doctor of Engineering (Dr.-Ing.)

Doctoral thesis by Markus Brandt

Darmstadt 2022

Assessors: Prof. Dr. Michael Waidner, TU Darmstadt
Prof. Dr. Haya Shulman, Goethe-Universität
Prof. Dr. Sebastian Schinzel, FH Münster

Department of
Computer Science

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Brandt, Markus: Bootstrapping Cryptography on the Internet
Darmstadt, Technische Universität Darmstadt,
Year thesis published in TUprints 2022
URN: urn:nbn:de:tuda-tuprints-215263

Date of submission: March 15, 2022
Date of viva voce: June 13. 2022

Darmstadt 2022

Please cite this document as:
URN: urn:nbn:de:tuda-tuprints-215263
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/21526

This document is provided by TUprints,
e-publishing service of TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

FÜR LENA.

Erklärungen laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Zusammenfassung

Diese Arbeit befasst sich mit dem Hochziehen (engl. bootstrapping) der Kryptografie, so dass ein theoretischer Algorithmus praktisch im Internet eingesetzt werden kann. Wir fassen die Anforderungen zusammen und definieren fünf Säulen, die die Grundlage für erfolgreich einsetzbare kryptografische Algorithmen bilden: Rechenleistung, Benutzerfreundlichkeit, Transport, Schlüsselverwaltung und Zufälligkeit. Wir konzentrieren und nur auf die drei letzten Säulen.

Für den Transport untersuchen wir zwei Hürden, die die Entwicklung praktischer, sicherer kryptographischen Anwendungen erschweren. Wir zeigen, wie wichtig die Auswahl von geeigneten Transportschichten ist. Unsere Auswertungen zeigen, dass das Transmission Control Protocol (TCP) die Bandbreite für Two Party Computation (2PC) Implementierungen nicht vollständig ausnutzt. Unsere Evaluation von drei Transportschichtprotokollen zeigt, dass kein Protokoll für jedes Szenario geeignet ist. Wir verwenden verschiedene Protokolle und Netzwerkbedingungen in mehreren geographischen Regionen, um die Auswirkungen auf die Leistung von 2PC Anwendungen zu verdeutlichen. Wir schlagen ein erweiterbares Framework vor, welches (zunächst) drei Transportschichtprotokolle integriert: User Datagram Protocol (UDP), TCP, und UDP-based Data Transfer Protocol (UDT). Die Aufgabe des Frameworks ist es, das am besten geeignete Transportschichtenprotokoll in Abhängigkeit von der aktuellen 2PC Anwendung und den Netzwerkbedingungen zu wählen.

Für die Schlüsselverwaltung zeigen wir, wie man einen Domänenvalidierungsmechanismus manipulieren kann. Wir haben einen optimierten BGP-Simulator entwickelt, um BGP-Pfade im Internet zu berechnen. Zusätzlich berücksichtigt unser Simulator die Beziehungen zwischen CAs in der Suche und vermeidet ungültige Pfade. In Kombination mit unserem Simulator analysieren wir die Widerstandsfähigkeit des

Domänen-Ökosystems gegenüber Angriffen auf die Domänenvalidierung. Unsere Evaluation zeigt, dass das Domänen-Ökosystem nicht widerstandsfähig gegen Präfix-Hijacks ist und dass nur wenige Autonome Systeme die meisten Domänen besitzen. Wir stellen mögliche Gegenmaßnahmen vor und schlagen die verteilte Domainvalidierung als Ersatz für die Standard-Domainvalidierung vor, welche einen starken Schutz gegen MitM-Angreifer bietet. Außerdem zeigen wir, dass viele IP-Adressen anycast sind, was für die verteilte Domainvalidierung von Vorteil ist. Wir analysieren die Platzierung der Validierungsagenten im Internet und demonstrieren eine Methode zur Bestimmung guter Autonome Systeme für die Agentenplatzierung.

Für die Zufälligkeit schlagen wir einen alternativen Ansatz zur Erzeugung pseudozufälliger Zeichenfolgen vor, welcher die verteilte Natur des Internets nutzt, um Zufall von öffentlichen Diensten im Internet zu sammeln. Wir stellen unseren Distributed Pseudorandom Generator (DPRG) vor und zeigen, die Sicherheit gegen starke Angreifer und wie Hauptmängel bestehender PRGs beheben werden. Der Generator basiert auf einer AES-Verschlüsselung im CBC-Modus und eine HDKF, um Zufälligkeiten und Eingaben für Handshakes zu extrahieren. Wir analysieren die Verteilung verschiedener Zufallsquellen wie HTTP, SMTPS, SSH und TOR und präsentieren eine Implementierung von DPRG unter Verwendung des TOR-Netzwerks. Wir analysieren die Qualität der Zufälligkeit und die Leistung unseres DPRG und zeigen, dass wir hochsichere Zufälligkeit erreichen können.

Abstract

This thesis focuses on bootstrapping cryptography, taking it from a theoretical algorithm to something we can use on the Internet. We summarize the requirement and define five pillars that build the foundation of successfully deployed cryptographic algorithms: computational performance, usability, transport, key management, and randomness. While there is a lot of research around the computational performance and usability of cryptographic algorithms, we focus on the other pillars.

For transport, we explore two obstacles that interfere with the development of practical, real-world secure computation applications. We show the importance of the selection of suitable transport layers. Our evaluations show how Transmission Control Protocol (TCP) does not fully utilize the bandwidth for Two Party Computation (2PC) implementations. We evaluate three transport layers protocols for different applications and show that no protocol is suited for every scenario. In our evaluations, we use various protocols and network conditions in multiple regions to highlight the effects on the performance of 2PC applications. We propose an extendable framework that integrates the (initially) three transport layer protocols: User Datagram Protocol (UDP), TCP, and UDP-based Data Transfer Protocol (UDT). The framework's task is to identify the most suitable transport layer protocol depending on the current TCP application and the network conditions.

For key management, we show how to manipulate a domain validation mechanism. We developed a BGP simulator to evaluate BGP paths on the Internet. Our simulator is high performant and respects relationships between CAs. In combination with our simulator, we analyze the resilience of the domains ecosystem to attacks against domain validation. Our measurements show that the domains ecosystem is not resilient to prefix hijacks and reveal that only a few ASes own most domains. We

discuss possible mitigations and propose the distributed domain validation as a drop-in replacement for the standard domain validation. It allows strong resistance against MitM attackers. Additionally, we show that many IPs are anycast which is beneficial for distributed domain validation. We also analyze the validations agents' placement on the Internet and demonstrate a method to determine good ASes for agent placement.

For randomness, we propose an alternative approach for generating pseudorandom strings, using the distributed nature of the Internet for collecting randomness from public services on the Internet. We develop our Distributed Pseudorandom Generator (DPRG) and demonstrate how it guarantees security against strong practical attackers and how it addresses the main shortcomings in existing PRGs. It uses AES encryption in CBC mode and an HDKF to extract randomness and inputs for handshakes. We analyze the distribution of different randomness sources like HTTP, SMTPS, SSH, and TOR and present an implementation of DPRG using the TOR network. We analyze the quality of randomness and performance of our DPRG and show that we can achieve highly secure randomness only from user space.

Publications and Contribution

Published:

[36] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, Domain Validation++ For MitM-Resilient PKI, In D. Lie, M. Mannan, M. Backes, and X. Wang (Eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018 (ACM, 2018).

[41] M. Brandt, H. Shulman, and M. Waidner, Internet As a Source of Randomness, In Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018 (ACM, 2018).

[38] M. Brandt, C. Orlandi, K. Shrishak, and H. Shulman, Transputation: Transport Framework for Secure Computation, In F. Kiefer and D. Loebenberger (Eds.) Crypto day matters 30 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[43] M. Brandt, H. Shulman, and M. Waidner, Internet As a Source of Randomness, In F. Kiefer and D. Loebenberger (Eds.) Crypto day matters 30 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[42] M. Brandt, H. Shulman, and M. Waidner, Distributed Domain Validation (DDV), In M. Selhorst, D. Loebenberger, and M. Nüsken (Eds.) Crypto day matters 31 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[39] M. Brandt, C. Orlandi, K. Shrishak, and H. Shulman, Optimal Transport Layer for Secure Computation, In Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020 (ICETE, 2020).

[40] M. Brandt and H. Shulman, Optimized BGP Simulator for Evaluation of Internet Hijacks, In 40th IEEE Conference on Computer Communications, INFOCOM 2021, Virtual Conference, Mai 11-13, 2021 (IEEE, 2021).

[37] M. Brandt, T. Dai, H. Shulman, and M. Waidner, Evaluating Resilience of Domains in PKI, In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Association for Computing Machinery, 2021).

My contribution

Scientific research is not only about the exchange of knowledge but also about collaboration. Many researchers discuss their research with other researchers to get feedback or new ideas. Usually, researchers concentrate their research on one research topic. When combining multiple different fields, a collaboration between researchers is inevitable. This thesis is no exemption to this. However, when research is collaborative, it is hard to break down contributions to each author. Other authors also participated in brainstorming, design, and interpretation of the results. So basically, they influenced each other.

Chapter 2, which is based on [38,39], was a joint work with Kris Shrishak and Claudio Orlandi. For me, it was a very close collaboration with Kris and our contributions merge almost seamlessly. Kris evaluated the performance of the protocols in our framework. I also contributed to the design of the framework and analysis of the transport layers. I designed and implemented the transport wrapper. Additionally, I included the wrapper into an existing framework for evaluation.

Chapter 3 is based on publications [36,37,40,42]. In [36], I did the design and implementation of the distributed domain validation tool. I also designed and implemented all the tools necessary for the evaluation of the defense mechanism and performed the evaluations of it. Tianxiang Dai and Amit Klein were responsible for the imple-

mentation and evaluation of the DNS cache poisoning attack. In the publications [37, 40,42], I was the only author except for my supervisors.

Chapter 4 is based on [41,43]. I was the only author except for my supervisors. I was responsible for all evaluations and implementations.

Contents

Zusammenfassung vii
Abstract ix
Publications and Contribution xi
1 Preface 1
1.1 Cryptography requirements 3
1.1.1 Computational Performance 3
1.1.2 Usability 3
1.1.3 Transport 4
1.1.4 Key management 4
1.1.5 Randomness 4
1.2 Structure of this work 5
2 Framework for Optimal Transport Layer for Secure Computation 7
2.1 Introduction 9
2.1.1 Contribution 11
2.1.1.1 Evaluations of Two Party Computation (2PC) applications 12
2.2 Background 13
2.2.1 Security levels 13
2.2.2 Oblivious Transfer 14
2.2.3 Garbled Circuits 15
2.2.4 Transport Layer Protocols 17
2.3 Related works 21
2.3.1 Garbling Circuit Optimizations 21
2.3.2 Secure Computation Frameworks 23
2.4 Framework 25
2.4.1 Two-Party Computation Layer 27
2.4.1.1 Garbling Schemes 28

2.4.1.2 Applications and Circuit Size 29
2.4.2 Transport Layer 30
2.4.2.1 Transport Protocols in our framework 31
2.5 Implementation 33
2.5.1 Abstraction 33
2.5.2 Simplification 34
2.5.3 Packet handling 35
2.5.4 Integration 36
2.6 Simulation 37
2.7 Evaluation 43
2.8 Conclusion 47
2.8.1 Future Research 47
3 Public Key Infrastructures 49
3.1 Introduction 51
3.1.1 Contribution 52
3.2 Background 55
3.2.1 Organization of Internet resources 55
3.2.2 Domain Name System 56
3.2.3 DNS cache poisoning 59
3.2.4 WHOIS 60
3.2.5 Domain Validation 60
3.3 Related Work 63
3.3.1 DNS cache poisoning 63
3.3.2 CA Compromises 64
3.3.3 PKI defenses 65
3.4 Off path attacks against Domain Validation 67
3.4.1 Triggering the DNS Request 67
3.4.2 Defragmentation Cache Poisoning 67
3.4.2.1 Forcing IP fragmentation 69
3.4.2.2 IPv4 Fragmentation Reassembly 70
3.4.2.3 Exploiting fragmentation 72
3.4.3 Overwriting DNS caches 73
3.4.3.1 Setup 74
3.4.3.2 Vulnerable Certificate Authorities 74
3.4.4 Challenges 75

3.4.5 Mitigations . . . 75
3.4.5.1 Blocking fragments . . . 76
3.4.5.2 DNSSEC . . . 76
3.5 Optimized BGP Simulation for Evaluations . . . 77
3.5.1 Implementation . . . 78
3.5.2 Correct interpretation of relationships . . . 78
3.5.3 Fast lookups . . . 80
3.5.4 Bidirectional Search . . . 80
3.6 Distributed Domain Validation . . . 83
3.6.1 Design and Implementation . . . 84
3.6.1.1 Agents . . . 84
3.6.1.2 Orchestrator . . . 84
3.6.1.3 Distributed Domain Validation . . . 85
3.6.1.3.1 Plain Distributed Domain Validation (DDV) . . . 85
3.6.1.3.2 BGP-aware DDV . . . 85
3.6.2 Evaluations . . . 86
3.6.2.1 Top-Level Domain TTL measurements . . . 86
3.6.2.2 Latency and failures . . . 86
3.6.2.3 Name server distribution . . . 88
3.6.2.4 Finding multiple paths . . . 90
3.6.2.5 Anycast IPs . . . 91
3.6.2.6 IP prefixes . . . 93
3.6.2.6.1 Sub-prefix hijacks . . . 94
3.6.2.6.2 Same-prefix hijacks . . . 96
3.6.2.7 Security evaluation of DDV . . . 97
3.6.2.8 Selecting agent ASNs . . . 98
3.7 Conclusion . . . 103
4 Randomness . . . 105
4.1 Introduction . . . 107
4.1.1 Randomness generation . . . 107
4.1.2 Contribution . . . 107
4.2 Background . . . 109
4.2.1 Randomness . . . 109
4.2.2 Cryptographic primitives . . . 110
4.2.3 TOR . . . 110

4.2.3.1 Consensus . . . 111
4.2.3.2 Keys and descriptors . . . 112
4.2.3.3 Circuits . . . 112
4.2.3.3.1 VERSIONS . . . 113
4.2.3.3.2 AUTH_CHALLENGE, AUTHENTICATE, and CERTS . . . 113
4.2.3.3.3 NETINFO . . . 113
4.2.3.3.4 CREATE, CREATE2, and NTOR handshake . . . 113
4.2.3.3.5 RELAY, EXTEND, EXTEND2, EXTENDED, and EXTENDED2 . . . 114
4.2.3.3.6 DESTROY . . . 115
4.3 Related Work . . . 117
4.4 Distributed Pseudorandom Generator . . . 119
4.4.1 Sources . . . 119
4.5 TORC . . . 125
4.5.1 Components . . . 125
4.5.2 Initialization . . . 126
4.5.3 Router selection . . . 127
4.5.4 Collecting randomness . . . 128
4.5.5 Randomness expansion . . . 129
4.6 Evaluation . . . 131
4.6.1 Router selection . . . 131
4.6.2 Quality of randomness . . . 131
4.6.3 Performance . . . 133
4.6.4 Security analysis . . . 135
4.6.4.1 At least one good server . . . 135
4.6.4.2 Hijacking attackers . . . 136
4.6.4.3 Reversing HKDF . . . 136
4.6.4.4 Untrusted local ISP . . . 137
4.7 Conclusion . . . 139
5 Summary . . . 141
6 Future Work . . . 143
A Appendix . . . 145
Bibliography . . . 149

Tables

2.1 Yao vs. GMW in LAN and intercontinental settings 27
2.2 Number of boolean gates 29
2.3 Garbled circuit sizes in MB 29
2.4 Experimental results for garbled circuits protocols 44
3.1 All 13 DNS root servers 57
3.2 List of supported DV methods of CAs 61
3.3 ASN statistics using CAIDAs dataset 78
3.4 Performance optimizations using direct memory lookups 80
3.5 Local vs. distributed resolution errors 87
3.6 Local vs. distributed resolution latency 87
3.7 Name server IP distribution by country 88
3.8 Name server IP distribution by RIR 88
3.9 IP address distribution by ASN 89
3.10 Top 20 ASNs for agent selection 99
3.11 ASNs from which name server went away 101
3.12 ASNs to which name servers changed 101
4.1 RIR distribution 120
4.2 Top 10 country distribution 121
4.3 Top 10 ASN distribution for Alexa 122
4.4 Top 10 ASN distribution for TOR 122
4.5 Statistical tests over sequences 132
4.6 Time to collect 1MB of random data 133

4.7 Time to collect 32 Bytes of random data 133
4.8 Time to collect different key sizes using a fixed bandwidth 134
A.1 Top 10 ASN distribution for HTTPS 146
A.2 Top 10 ASN distribution for SMTP(S) 146
A.3 Top 10 ASN distribution for SSH 147

Figures

1.1 Pillars of deployable cryptography . . . 3
2.1 RSA implementation of 1-2 Oblivious Transfer . . . 14
2.2 Overview of evaluating a function using garbled circuits . . . 15
2.3 AND gate used in Garbled Circuits . . . 16
2.4 Garbling steps . . . 16
2.5 Positioning our work within related work . . . 21
2.6 Overview of our framework . . . 25
2.7 Effect of latency on a 10Gbps link . . . 38
2.8 Effect of loss on a 10Gbps link . . . 39
2.9 Performance of SHA256 using GLNP15 . . . 40
2.10 Performance of a JustGarble protocol . . . 41
2.11 Performance of a GLNP15 protocol . . . 41
2.12 Comparison of assumptions between EU and AUS . . . 45
2.13 TCP vs UDT as garbled circuit size increases . . . 45
3.1 DNS request packet . . . 68
3.2 ICMP fragmentation indicating an MTU of 68 bytes . . . 70
3.3 Malicious second fragment modifying the mail server . . . 72
3.4 First fragment from the name server to the resolver . . . 73
3.5 AS cashflow . . . 78
3.6 Valid and invalid paths . . . 79
3.7 Node expansion for the path search . . . 81
3.8 CDF for name server distribution . . . 89

3.9 Number of ASNs for each name server 90
3.10 Number of unique AS paths per domain 91
3.11 Round-trip time triangulation 92
3.12 Average ping from all vantage points 92
3.13 Domains vulnerable to sub-prefix hijacks 94
3.14 Distribution of domains to prefixes 95
3.15 IP prefix statistics 95
3.16 Success probability simulation 96
3.17 Hops in-between Let's Encrypt vantage points and name server ASes . 97
3.18 Simulating attack success rate 97
3.19 Simulation of the success rate of an attacker 98
3.20 Simulating attack success rate of top 10 100
4.1 Chain of trust 111
4.2 IP to ASN distribution 121
4.3 TORC circuit creation 128
4.4 First router selection while bootstrapping 131
4.5 Visualization of collected randomness using TORC 132
4.6 Simulation of the success rate of an attacker 135

Listings

2.1 Example server setup in plain C/C++ 34
2.2 Example using our wrapper 35
2.3 Example of an echo client using our wrapper 36
3.1 Example zone file for example.org 56
4.1 Example torc.conf configuration file 125

Algorithms

4.1 Initialize cipher 126
4.2 Random router selection 127
4.3 Get random string from router 129
4.4 Expand randomness 129

Abbreviations

2PC	Two Party Computation
AES	Advanced Encryption Standard
API	Application Programming Interface
AS	Autonomous System
ASN	Autonomous System Number
BGP	Border Gateway Protocol
CA	Certificate Authority
CAIDA	Center for Applied Internet Data Analysis
CBC	Cipher Block Chaining
CDF	Cumulative Distribution Function
CDN	Content Distribution Networks
CPLD	Complex Programmable Logic Device
CRL	Certificate Revocation List
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
CSR	Certificate Signing Request
CT	Certificate Transparency
DDoS	Distributed Denial of Service
DDV	Distributed Domain Validation
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DPRG	Distributed Pseudorandom Generator
DV	Domain Validation
ECC	Elliptic Curve Cryptography
EV	Extended Validation
FPGA	Field-Programmable Gate Array
GUI	Graphical User Interface

HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HMAC	Hash-based Message Authentication Code
HPKP	HTTP Public Key Pinning
HSTS	HTTP Strict Transport Security
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
LIR	Local Internet Registry
MitM	Man in the Middle
MPC	Secure Multi-Party Computation
MTU	Maximum Transmission Unit
OCSP	Online Certificate Status Protocol
OT	Oblivious Transfer
OV	Organization Validation
PCC	Performance-oriented Congestion Control
PKI	Public Key Infrastructure
PPPoE	Point-to-Point Protocol over Ethernet
PRF	Pseudorandom Function
PRG	Pseudorandom Generator
PRNG	Pseudorandom Number Generator
RIR	Regional Internet Registry
RNG	Random Number Generator
RSA	Rivest–Shamir–Adleman
RTT	Round-Trip Time
SABUL	Simple Available Bandwidth Utility Library
SCSV	Signaling Cipher Suite Value
TCP	Transmission Control Protocol
TLD	Top-Level Domain
TRNG	True Random Number Generator
TTL	Time to live
TXID	Transaction ID
UDP	User Datagram Protocol
UDT	UDP-based Data Transfer Protocol

URL	Uniform Resource Locator
VM	Virtual Machine
WAN	Wide area network

CHAPTER 1
Preface

Cryptography is crucial for our modern world. Without it, we could not use the technologies we rely on every day. Many researchers work on improving cryptography. They make it faster, more secure, or invent new algorithms. However, this research focuses on the algorithms. There are many requirements for cryptographic algorithms to be of practical use. If we do not meet them, the best and most secure algorithm becomes useless.

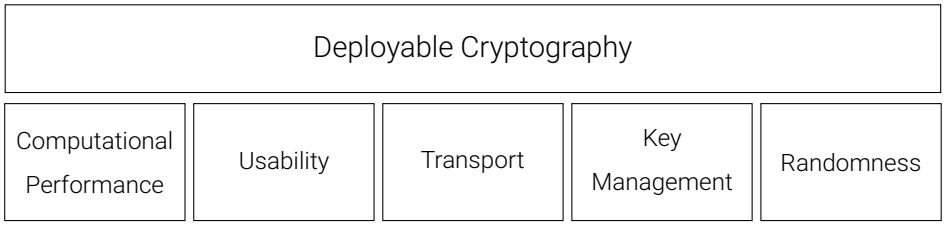


Figure 1.1 Pillars of deployable cryptography

1.1 Cryptography requirements

In this thesis, we want to focus on what it takes to bootstrap cryptography, taking it from a theoretical algorithm to something we can use on the Internet. We usually delay the deployment of new algorithms until we can gather more experience and assure the algorithm is proven secure. But even if it is, there might be obstacles to practical deployment. Depending on the type of algorithm, it may need, e.g., specific keys or require a lot of bandwidth. We summarize the requirement, as shown in Figure 1.1, and define five pillars that build the foundation of successfully deployed cryptographic algorithms: computational performance, usability, transport, key management, and randomness.

1.1.1 Computational Performance

For an algorithm to be usable, it must be performant. Encryption or decryption must finish in a reasonable time and work with limited CPU and RAM resources. Depending on its use case, the algorithm must be suited to smaller devices. If we want to use an algorithm for mobile phones or even IoT devices, the devices must have sufficient resources to use these algorithms. Researchers and programmers put in a lot of effort to make algorithms as performant as possible. There exists so much work in that field that we do not must include it in our research. We can assume that algorithms are or will be performant.

1.1.2 Usability

Usability is a research field that is out of our scope. Algorithms that are too complex or too hard to implement are dangerous. Users might avoid it if it is too complicated. Also, if they do not fully understand it, they might implement it incorrectly and generate vulnerabilities. In both cases, it makes a successful practical deployment impossible.

1.1.3 Transport

Algorithms usually build on cleanroom design. The creators assume perfect bandwidth conditions. Or they design theoretical algorithms which do not include any transport layers. Depending on the network conditions, transport layers can play a big part. We investigate the importance of suitable transport layers in Chapter 2. We show that under certain network conditions, the correct transport layers make otherwise unusable cryptography usable.

1.1.4 Key management

Some algorithms require asymmetric keys, others symmetric keys. Some may have short keys, e.g., 256 bits others have keys over 1 megabyte, e.g., Code-based McElise crypto. Key distribution is no easy task, especially when assigning new keys to objects through ownership verification. One example is the worldwide web. To hand out new certificates, which bind keys to domains, we have to verify that the request comes from the legitimate owner of that domain. In Chapter 3, we show the weaknesses of this procedure. We demonstrate attacks and provide a detailed evaluation and countermeasures.

1.1.5 Randomness

For all kinds of cryptographic algorithms, we need randomness. We could say that randomness is the most crucial building block for cryptography. If we can predict keys and passwords, even the most secure algorithm will not be safe. Cryptographic publications often assume that randomness is ubiquitous, and we can also create cryptographically secure random strings. However, randomness collection is not a trivial task. For cryptography, we often use pseudo-random sequences with predefined properties. However, algorithmic random generation is deterministic and can be dangerous if seed values are not random. In Chapter 4, we present a new way of collecting randomness using servers on the Internet. We show how to generate cryptographically secure random strings on hardware with no entropy.

1.2 Structure of this work

We investigate three requirements for deployable cryptography. For every pillar, we made a single chapter. Even if this thesis is monographic, we want to allow readers to skip individual chapters. Every chapter describes its background necessary to follow the rest of the chapter. We want to split up the complex topic of bootstrapping cryptography on the Internet into smaller chunks. However, since it is a monographic work, the preface, summary, and future work include every chapter.

CHAPTER 2

Framework for Optimal Transport Layer for Secure Computation

In this chapter, we will investigate the importance of transport layers for cryptography. We will see how transport layers affect the usability of more complex cryptographic protocols. To demonstrate this, we have chosen Two Party Computation (2PC).

2.1 Introduction

2PC is a sub-problem of Secure Multi-Party Computation (MPC), also called Secure Computation, MPC, or Privacy-Preserving Computation, with only two parties. In 2PC, two parties jointly compute a function over their private inputs without revealing these.

The need for secure and efficient 2PC solutions is increasing, and many applications would benefit from 2PC. A classic scenario for 2PC would be two parties that want to perform functions on their combined data. However, they do not trust each other or reveal any inputs. Achieving this might not be possible without 2PC due to privacy concerns, competition (e.g., financial), or legislation. An often-used example to illustrate this problem is the private set intersection, where a secret service holds a list of terrorists, and an airline has a list of passengers. Both parties do not want to reveal any information besides the intersection itself. With 2PC, it is possible to realize previously impossible solutions for applications like key management for digital currencies [12], auctions [34], tax-fraud detection [33], private set intersection [100, 133] and even prevention of satellite collision [92]. With the rise of privacy awareness and privacy laws, 2PC will become an important method for maintaining privacy.

However, 2PC, and MPC in general, currently have drawbacks. One of them is that implementations are still not practical. This field focuses on theoretical research applications or libraries that resemble a Proof-of-Concept to demonstrate the feasibility [88]. Evaluations use simulations or single hosts without respect to real-world scenarios with realistic network conditions (e.g., including latency and packet loss) [115]. Current implementations are far from being usable, which leaves users with a tradeoff between privacy and efficiency, usually by relying on third parties.

Another drawback is that computational speed was the main goal for 2PC. The idea of 2PC by Yao in 1986 [164] is quite old. With the first public implementation of 2PC in 2004 [121], it has seen huge improvements. The most efficient implementations built upon Garbled Circuits [24] and Oblivious Transfer (OT) [126]. Most improvements come from protocol optimizations, hardware-accelerated cryptographic operations, and multiple CPU cores. These drastic improvements lead to the belief that we reached the lower limit for current 2PC implementations [166]. Due to these advanced computation optimizations, the common assumption is that *bandwidth* is the only

remaining bottleneck [16,166]. Work to reduce the complexity and amount of data the parties must exchange, effectively improving bandwidth, exist. However, no research evaluates the network layers of 2PC. While evaluations on a single machine yield a practical performance [23,139], evaluating on different hosts, the overhead in bandwidth and latency make it unsuitable for practical use [129,157]. These isolated evaluations ignore issues that occur in the practical use of 2PC like latency, traffic bandwidth, packet loss, and congestion which can affect real-world performance [88, 115].

2.1.1 Contribution

We explore two obstacles that interfere with the development of practical, real-world secure computation applications. First, it should be easy to integrate different transport layers, and second, there should be an automated way to evaluate and compare the performance of 2PC implementations in different network setups.

We show in our evaluations how Transmission Control Protocol (TCP) does not fully utilize the bandwidth for 2PC implementations. High latency and packet loss cause a degrading performance of TCP. It also fails to adapt to rapidly changing network conditions. In stable network conditions, TCP also does not yield optimal performance making it unsuitable for different 2PC applications. However, all 2PC implementations use the standard TCP socket provided by their operating systems.

Different variants of TCP and other transport protocols showed up but are still unused by 2PC applications. There are reasons for this. E.g., to change a transport layer like TCP, one must include the changes into the OS kernel. Of course, user-space transport layer protocols do not need this. But these protocols are still niche protocols, and many users, including 2PC developers, have not heard of them. That is why we propose a framework that automatically switches to the best transport layer for the given 2PC application.

Using our framework, we demonstrate the improvements which we can achieve by using other transport layers. We integrated three transport layer protocols: User Datagram Protocol (UDP), TCP, and UDP-based Data Transfer Protocol (UDT). We chose UDT because of its increasing popularity and its impressive performance. The framework identifies the most suitable transport layer protocol depending on the current TCP application and the network conditions. An adaptive behavior like this allows us to achieve higher performance and throughput on complex real-world networks with various network conditions. In our evaluations, we use different protocols and network conditions in multiple regions to highlight the effects on the performance of 2PC applications.

2.1.1.1 Evaluations of 2PC applications

Creating real-life testbed environments is difficult and troublesome. 2PC developers usually avoid this and evaluate their implementations on single hosts or simulated environments [88, 115]. These environments, however, are not representative of real-world networks like the Internet. They miss situations caused by various network conditions. There are other platforms to perform experiments in distributed setups [52]. However, these are not suited for 2PC evaluations and do not support non-standard transport layers.

2PC developers would have to set up and install the requirements. These requirements include uploading binaries, installing libraries and dependencies, integrating transport layer protocols, measuring latencies and packet loss.

We developed a distributed 2PC testbed using our preliminary framework. The testbed provides a user-friendly Graphical User Interface (GUI) where users can select the application and specific network conditions, e.g., simulated packet loss, delay, bandwidth, or using geographically distributed servers. The testbed's infrastructure uses Vultr VMs, which we can setup in various parts of the world. Using our testbed 2PC, developers can perform real-life evaluations and receive immediate results without the need to install or use any traffic monitoring tools.

Using our platform, we show that even general-purpose protocols, like UDT, provide a significant performance improvement over standard TCP sockets. E.g., UDT performs eight times better than TCP for large circuit sizes in WAN settings. Custom transport layer protocols completely designed to optimize 2PC applications would, of course, improve the performance even more. We can see that research on optimizations for specific tasks, e.g., [8,45,86,130,161] exists. Optimizations specifically for crypto applications like 2PC are missing.

2.2 Background

MPC is a complex subject. A complete background would exceed the scope of this thesis. Therefore, we will only cover the basics needed to explain our contribution. Also, we will only focus on 2PC when we are talking about MPC̀

2.2.1 Security levels

MPC protocols must be secure against attackers. Mathematical proofs are the method of choice to prove security properties. Part of these definitions is the environment and the adversaries the protocols face. These are the common categories for adversaries:

Semi-Honest

For semi-honest, we assume a passive adversary. Parties follow the protocol and try to extract information out of the protocol. Protocols achieving this security level are efficient and prevent accidental leakage of information. These protocols provide a weak level of security which is often an initial step to achieve a higher security level.

Malicious

The malicious adversaries are active adversaries that do not necessarily follow the protocol. It may manipulate messages and violate the protocol to extract additional information. Protocols achieving this level provide high security. If these parties build a majority, the only thing these parties can achieve is to abort the protocol.

Covert

Covert adversaries are somewhere in-between active and passive adversaries. They represent more of a real-world model than the other two. At this security level, adversaries can cheat but also have a high probability of "getting caught". Thus, adversaries are less likely to cheat if others can expose it. Aumann and Lindell introduced covert adversaries in 2007 [17].

In our research, we focus on semi-honest protocols. Researchers improved the efficiency of these protocols to a level where the communication layer is the bottleneck [16,166].

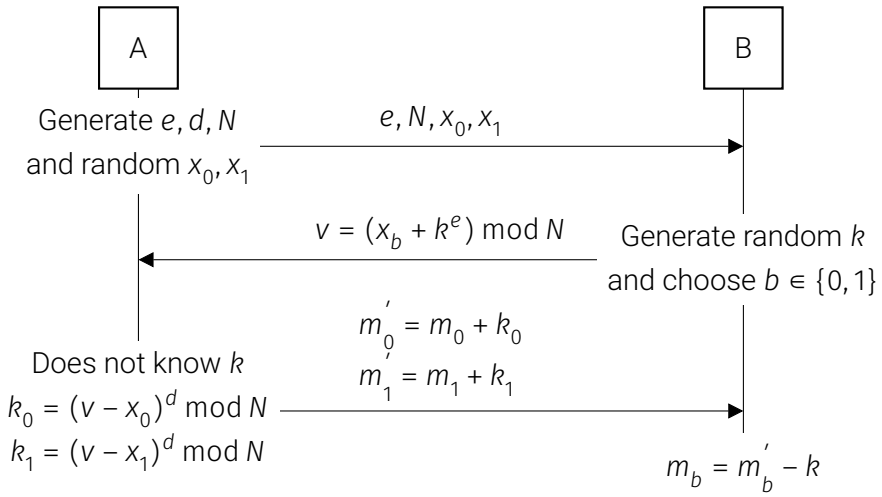


Figure 2.1 RSA implementation of 1-2 Oblivious Transfer based on the protocol by Even, Goldreich, and Lempel

2.2.2 Oblivious Transfer

In the OT protocol, a sender can send multiple messages to a receiver, but the receiver can only choose a certain amount (k) of all (n) messages. Additionally, the receiver cannot gather any information about the messages it did not choose. This setup is a k-out-of-n oblivious transfer. In Figure 2.1, we see an example of a one-out-of-two (1-2) oblivious transfer using RSA cryptosystem. In this example, the receiver gets two messages and can only read one message. To do this, the sender (A) will create an RSA keypair and two random messages, and the receiver (B) will choose one of the two random messages, add its random k to it, and send it back to A. Because A does not know k, it does not know which message B chose. A will generate two ks (k_1, k_2) and add each to one message it wants to send. Since B knows the correct k, it can decrypt the message but cannot decrypt the message it did not choose.

For OT, we can use different public-key cryptographies, such as RSA, Diffie-Hellman, Elliptic Curve Cryptography (ECC) or post-quantum cryptography. Because public-key cryptography is computationally very costly compared to symmetric cryptography, 2PC applications use OT extensions [16,104], which use symmetric cryptography operations. However, OT implementations are abstract, and a user does not have to know its internal protocol to use it.

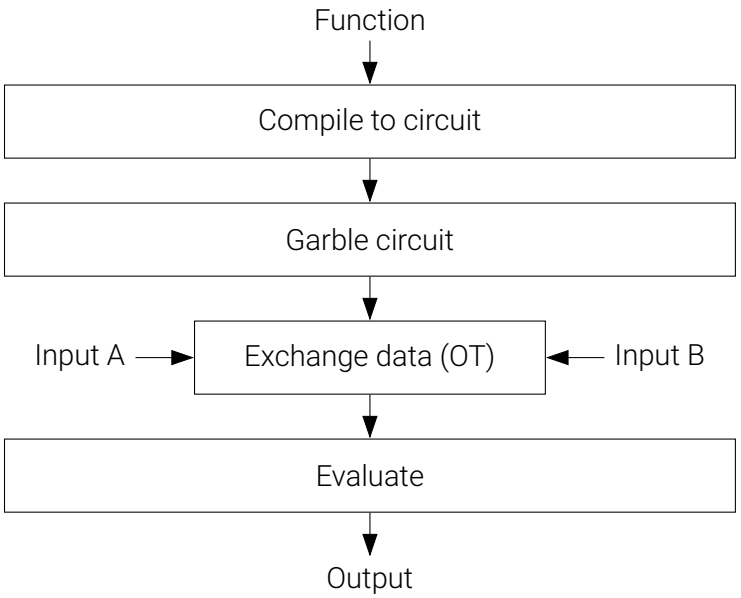


Figure 2.2 Overview of evaluating a function using garbled circuits

2.2.3 Garbled Circuits

Garbled circuit is a cryptographic protocol that enables parties to compute a function without revealing their inputs. For this, we transform any function we want to evaluate into a Boolean circuit. Commonly used Boolean gates are XOR and AND gates. This process is like the design of circuits for Field-Programmable Gate Arrays (FPGAs) or Complex Programmable Logic Devices (CPLDs). These usually only support one type of gate because of their hardware design. We use these circuits as our initial step for 2PC.

We show a simplified overview in Figure 2.2. The Garbled Circuit protocol will evaluate the circuits using cryptographic functions. We can process multiple Gates in parallel if they do not depend on the output of previous gates. To evaluate a single logic gate, we use cryptographic keys to represent the logical values 0 and 1.

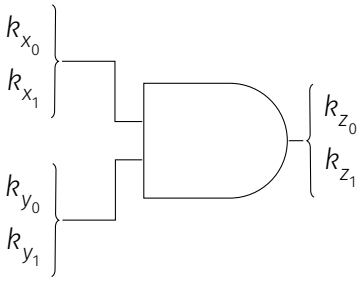


Figure 2.3 AND gate used in Garbled Circuits

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(1)

$\xrightarrow{garble}$

$\text{Enc}_{k_{x_0},k_{y_0}}(k_{z_0})$
$\text{Enc}_{k_{x_0},k_{y_1}}(k_{z_0})$
$\text{Enc}_{k_{x_1},k_{y_0}}(k_{z_0})$
$\text{Enc}_{k_{x_1},k_{y_1}}(k_{z_1})$

(2)

$\xrightarrow{shuffle}$

$\text{Enc}_{k_{x_1},k_{y_0}}(k_{z_0})$
$\text{Enc}_{k_{x_1},k_{y_1}}(k_{z_1})$
$\text{Enc}_{k_{x_0},k_{y_1}}(k_{z_0})$
$\text{Enc}_{k_{x_0},k_{y_0}}(k_{z_0})$

(3)

Figure 2.4 Garbling steps

Let us assume the AND gate in Figure 2.3. In this example k_{x_0} represents $x = 0$ and k_{x_1} represents $x = 1$. The same applies to the y input and its keys. When we input k_{x_1} and k_{y_1} into the AND gate we will receive k_{z_1}. For any other combination, we will receive k_{z_0}. We achieve this by garbling the truth table as seen in Figure 2.4. First, one party garbles the truth table (1) by encrypting the output key z with its corresponding x and y keys. E.g., $\text{Enc}_{k_{x_0},k_{y_0}}(k_{z_0})$ means, that we encrypt k_{z_0} with the keys k_{x_0} and k_{y_0}. By doing this, we get four encrypted keys (2) where each combination of x and y can only decrypt one entry. If we keep the order, we will leak information about the inputs. E.g., one party receives the garbled truth table and an unknown x key. If we chose y to be 1 and see that we can decrypt the last value, we know that x must be 1. To prevent this, we shuffle the order of the items (3), so we cannot extract any information from their positions. So, for each gate, party A sends the fully garbled output keys and its x key party B. Party will exchange the key for y by using OT. That way, party A does not know the logical value of input y and output z, and party B does not know the logical value of x. Party will use z as the next input or reveal it processed the whole circuit.

2.2.4 Transport Layer Protocols

Transport layer protocols are essential for transmitting data between hosts. Their main task is to provide the functionality to send messages with variable lengths from one host to another. Besides this bare minimum, most protocols include additional features for reliability. These include resending missing packets, dropping duplicate packets, reordering out-of-order packets, and congestion control. If the data is longer than the maximum packet size of the underlying network layer, hosts must split the data into smaller chunks that fit. Congestion control is necessary then because one must time the packets while sending so the receiver's buffers do not overflow because it is too slow. When sending too fast, the sender can also overflow its send buffer. However, not all transport layer protocols support this. The choice of the best transport layer protocol strongly depends on the application. This dependency makes them an essential part when designing networked applications.

Congestion control is a complex topic, and there are multiple approaches for improvements [45,46,86,120,158,160]. However, when violating their network assumptions, these improvements do not perform consistently. Most congestion control algorithms rely on either packet loss or packet delay for their measurements. We will talk about these algorithms when we describe TCP and UDT.

The UDP is a very minimalistic transport layer designed for optimal networks and small data packets. It does not feature any duplicate packet or packet loss detection. If datagrams (UDP packets) arrive out-of-order, it will not reorder them. There is also no congestion control. When the host sends too fast, datagrams get lost or dropped. If one needs these features, one must implement them on the application layer. We will see examples of this later. Without any of these features, plain UDP is unsuitable for 2PC since reliance and congestion control are indispensable for cryptographic protocols.

The TCP is the most used transport layer protocol. It provides reliance and congestion control. Like UDP, all common operating systems include it in their kernels, making it available to everyone out-of-the-box. One of the most used congestion control algorithms for TCP is CUBIC [86]. It is a loss-based congestion control algorithm that uses a cubic function to manage congestion control. This cubic function consists of two parts: (a) a concave portion where the window size quickly rises to the size before the last loss and (b) a convex growth where CUBIC tries to increase the bandwidth,

starting slowly and rapidly increasing afterward. CUBIC defines the congestion window size with the equations

$$cwnd = C(T - K)^3 + w_{max}, \tag{2.1}$$

$$K = \sqrt[3]{\frac{w_{max}(1-\beta)}{C}}, \tag{2.2}$$

where C is a scaling constant defined as 0.4 (RFC 8312), β is a multiplicative decrease factor defined as 0.7 (RFC 8312), T is the elapsed time since the last window reduction, and w_{max} is the window size before the latest size reduction. When further talking about TCP, we will assume TCP-CUBIC.

The UDT is a user-space protocol that manages congestion control and reliability in user-space. In contrast, other protocols usually implement these functions into the kernel of the operating system. User-space code has limitations because, unlike kernel-space code, it cannot directly interact with parts of the operating system and must do that using system calls. However, it has the advantage that we do not have to modify components of the operating system to implement changes. The main goal is high throughput for large datasets over high-speed connections (e.g., 1 Gbit/s). UDT uses timer-based selective acknowledgments (ACK) and packet-based sequencing. Negative acknowledgments (NAK) indicate a packet loss. It uses a hybrid rate-based congestion control and window-based flow control. Rate control, which manages packet sending rate, triggers at a constant interval (every SYN), while window control, which limits the number of unacknowledged packets, triggers when receiving an acknowledgment packet. The packet sending rate is an additive increase and multiplicative decrease algorithm. The multiplicative decrease is by a factor of 1/9, while the additive increase is independent of the RTT. For every rate control interval, if there is no negative feedback, the packet-sending rate x increases by $\alpha(x)$. The definition of $\alpha(x)$ is

$$\alpha(x) = 10^{\lceil \log(L - C(x)) \rceil - \tau} \cdot \frac{1500}{S} \cdot \frac{1}{SYN} \tag{2.3}$$

where x has the unit of packets/second, L is the link capacity measured by bits/second, S is the packet size (in terms of IP payload) in bytes, $C(x)$ is a function that converts the unit of the current sending rate x from packets/second to bits/second ($C(x) = x * S * 8$), and $\tau = 9$ is a protocol parameter.

There are other user-level protocols like QUIC or Performance-oriented Congestion Control (PCC). We only chose TCP and UDT for our research because other protocols were not easy to integrate at the time. E.g., QUIC only had a reference implementation as an HTTP server and was not usable as a general-purpose transport layer and PCC, which is a further improvement to UDT, is only unidirectional. Also, we did not investigate other TCP congestion controls protocols since modern operating systems only support very few, with CUBIC being the best option.

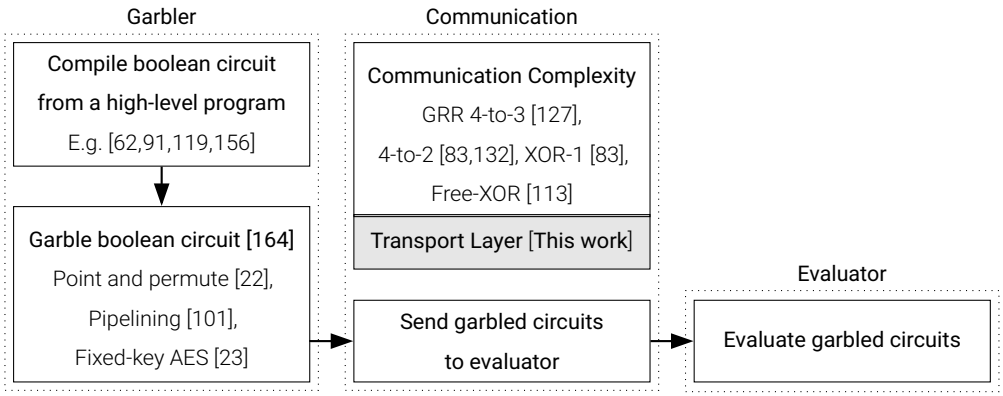


Figure 2.5 Positioning our work within related work

2.3 Related works

Figure 2.5 graphically illustrates the relationship of our work compared to other related work. The Secure Computation field is a broad and diverse topic. There are various protocols for a different number of parties, adversarial models, corruption thresholds. We, however, decided to focus on the most natural case of Secure Two-Party Computation and leave the research of multi-party protocols as future work.

2.3.1 Garbling Circuit Optimizations

The original protocol for garbling circuits by Yao [164] dates to 1986. Each Boolean gate requires four ciphertexts and four decryptions. Since then, many works have tried to improve the efficiency of garbled circuits by reducing the number of keys or cryptographic operations. The Point-and-Permute strategy [22] reduced the number of decryptions per gate to one. Also, it divides the size of ciphertext by approximately two.

Communication Complexity, the most important measure, is the number of ciphertext transmissions needed per Boolean gate. The 4-3 GRR (garbled row reduction) technique [127] first reduced the number of ciphertexts to three per Boolean gate. Furthermore, the 4-2 GRR [132] reduced the number of ciphertexts needed even further to only two. The Free-XOR technique [113] further optimized the efficiency. Using this technique, XOR gates (or any other linear gate) do not require the transfer of ciphertext. The Free-XOR and 4-2 GRR, however, were incompatible with each other.

The Half-Gate optimization [166] combines the benefits of Free-XOR and 4-2 GRR. However, the Free-XOR technique is only secure in the Random Oracle Model [25]. It requires a form of circular security assumption [49].

Bellare et al. [23] proposed faster garbling schemes using stronger assumptions, like Fixed-Key block ciphers. However, some criticize achieving efficiency at the cost of stronger security assumptions. The industry usually adopts conservative approaches since it is challenging to change protocols after vulnerabilities emerge after deployment. Consequently, [83] proposed a proven secure construct for garbled circuits which only uses standard assumptions. It requires two ciphertexts per AND gates (like 4-2 GRR) and one ciphertext for each XOR gate (XOR-1). Gueron et al. [83] concluded that the cost for higher security guarantees is much smaller in practice than in theory. Our evaluations support this conclusion since we show that choosing an optimal transport layer has a much higher impact than betting on non-standard assumptions to boost efficiency. E.g., using UDT and standard assumptions over WAN is four times faster than TCP with non-standard assumptions.

Using multiple parallel TCP connections, Nielson, Schneider, and Trifiletti [129] tried to improve efficiency. By doing so, the saturation of the network bandwidth is higher. However, an arbitrary number of parallel connections can lead to network congestion effectively, lowering throughput [87].

2.3.2 Secure Computation Frameworks

Since the introduction of the Fairplay framework [121], development on various other 2PC frameworks started [62,73,101]. Libscapi [73] provides a garbled circuit framework with the latest optimizations with security against active and passive adversaries. ABY [62] is a framework for mixed protocols and combines arithmetic sharing, Boolean sharing, and garbled circuits. These frameworks only focus on the secure computation aspect and rely on plain TCP sockets for communication.

[91] surveys general-purpose compilers for secure computation used to provide high-level abstractions to describe functions in an intermediate representation (such as a circuit). While [91] focuses on compilers, we focus on the efficiency of protocol execution. The examples chosen in [91] demonstrate the usability of the compiler code rather than representing practical MPC use cases. Also, the tests run in a standalone environment and do not account for networks conditions. However, the paper has a different goal. It wants to demonstrate the efficiency of the protocols. There is no intention to provide a practical testing framework.

In [19], the authors consider providing MPC as a service. Users can use a platform to run their protocols through a web browser of a phone app. Their research focuses on low-bandwidth MPC protocols, which require multiple rounds in LAN settings. Our work concentrates on high-bandwidth constant round 2PC protocols in LAN and WAN settings.

We can see how far the efficiency in terms of computational complexity has progressed. In general, we achieved a state where the evaluation on a single host yields satisfactory performance. However, network conditions can impact the overall performance. No previous work considered adjusting transport layers to improve the performance of Secure Computation protocols.

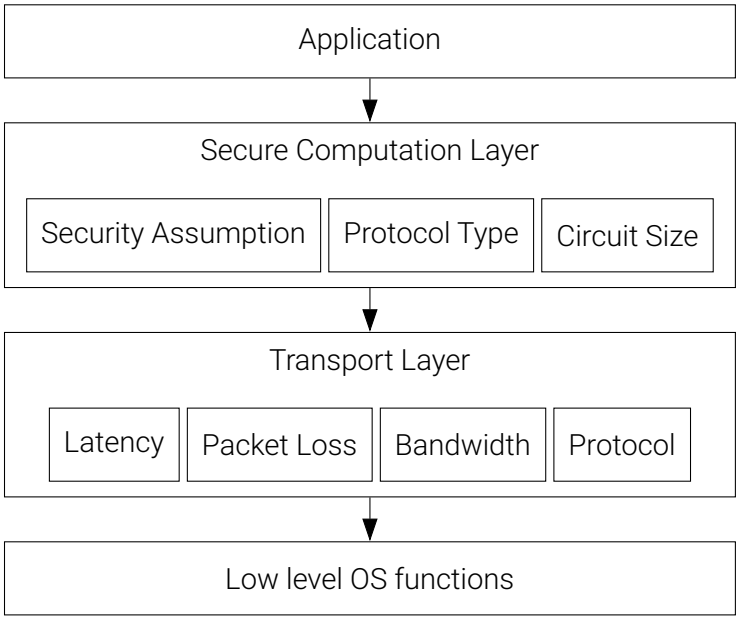


Figure 2.6 Overview of our framework

2.4 Framework

We propose a framework to simplify the use of different transport layers in secure computation. Our framework has two goals: identify which transport layer protocol is optimal for the current task, and isolate components by function. It uses the input sizes, the network setup, and buffer sizes to decide which transport layer it uses. To separate parts of the application, it enforces two layers (see Figure 2.6). One layer is the secure computation layer. It accepts the function for evaluation and the size of the circuit representing this function. With the given parameters, we can deploy optimizations relevant for performance improvements on the transport layer. The other layer is responsible for the abstraction of network connections. This layer provides a general abstract API. Every component can use it without knowing how it works and which protocol it is currently using. The layer determines the latency, bandwidth, and packet loss on the network through live probes. The transport layer is also responsible for avoiding overflow at the sender and the receiver.

We demonstrate the usage of the framework and the evaluation of different 2PC implementations. We integrated popular and representative protocols into the framework in both the secure computation and the transport layer. In the secure computation layer, we integrated a garbled circuit protocol under standard assumptions [83],

a garbled circuit protocol under circularity assumption [166], and the scheme that assumes AES as an ideal cipher [23]. For the transport layer, we support UDP, TCP [86], and UDT [82].

Our framework bridges the secure computation layer and the transport layer, choosing a suitable protocol for its given parameters. Independent extension of the layers is also easy. One can easily integrate new protocols in either layer. This flexibility is crucial because research for new protocols is still continually active. Also, old protocols still get improvements. Future protocols may provide a better solution than current options.

We have implemented abstract classes into our framework, making the selection of transport layer protocols simple or even automated. Integrating new protocols into an existing 2PC application is straightforward. Once the protocol is available to the framework (e.g., through an update), the user must define a string with the name of the new protocol to use it manually.

Without a framework like this, one must rewrite large chunks of the secure computation's code. Additionally, the rewrite can be especially hard since many existing Secure Computation implementations do not have a modular design. The network code mixes with the application code, and the transport layer protocol is hard-coded into the application code. Often, IP addresses and port numbers are hard-coded. Nowadays, most Secure Computation implementations are only a proof-of-work and not a production-ready solution [88]. Uncluttering these and adding transport layer protocols requires an extensive rewrite of the codebase.

There are two ways developers can use our framework. First, they can use the transport layer module to select the best-suited transport protocol automatically. The transport layer, together with the secure computation layer, will choose the most efficient settings. Second, the developer can use the secure computation layer separate from the transport layer and manually evaluate different transport protocols.

Circuit	Setting	Yao (GLNP15)	GMW
AES	LAN	7.00	27.88
	EU-US	130.40	2917.30
	EU-AUS	377.92	7325.00
SHA1	LAN	14.50	1651.27
	EU-US	144.41	187080.60
	EU-AUS	396.52	493143.10

Table 2.1 Yao vs. GMW in LAN and intercontinental settings (runtime in ms)

2.4.1 Two-Party Computation Layer

The most efficient implementations for 2PC use Yao's protocol with garbled circuits. It utilizes asymmetric operations with only a few computationally expensive calculations like exponentiations. The GMW protocol [80] is the main alternative to Yao's protocol. It uses only OT, and unlike Yao's protocol, it does not have constant rounds. Overall, as seen in Table 2.1, GMW is less efficient than Yao. That is why we do not consider it for our research.

We focus on recently optimized garbling schemes for which we consider three main characteristics. For one, the size of the produced circuits, which affects the efficiency of the communication. Then, the number of cryptographical operations necessary for generating and evaluating a garbled circuit. These influence computational efficiency. And finally, the security assumptions, which prove the security of the scheme. We included the most representative candidates, with the best efficiency-security trade-off, into our framework. Our evaluations extend upon the work of GLNP15 [83]. We show that we can achieve a more efficient secure computation with standard assumptions when selecting better transport layer protocols.

2.4.1.1 Garbling Schemes

For the garbled circuits, we use the implementations of libscapi [73]. It incorporates all known optimizations and uses AES-NI.

GLNP15 This implementation uses the garbling scheme described in [83]. Its main advantage is that it only makes conservative computational assumptions, i.e., we can prove it secure under the assumption that AES behaves like a pseudorandom function (PRF). It garbles linear gates (e.g., XOR) and non-linear gates (e.g., AND) differently. Using the 4-2 Garbled Row Reduction (GRR), Garbling an AND gate produces two ciphertexts. Garbling an XOR gate, using the XOR-1 technique, produces only one ciphertext.

Half-Gates The second implementation builds on the work of [166]. It uses the so-called Half-Gate optimization, which is compatible with the Free-XOR optimization described in [113]. However, this optimization requires a stronger computational assumption on AES. It is complex, and we refer for details to [166].

JustGarble The final implementation we consider originates from the JustGarble framework proposed by [23]. Key-scheduling is the most expensive phase when using the AES-NI. AES-NI performs best when a large amount of encrypted data using the same key. When keys consistently change, AES-NI performs worse. However, garbling circuit schemes use different keys for each gate. JustGarble uses AES in stream cipher mode as an ideal permutation. The key is a fixed constant. To encrypt message m using the key k one computes $C = AES_c(k) \oplus m$ for some constant c. This non-standard usage of AES. However, out of the three presented garbling schemes, it has the best efficiency/security trade-off.

Function	AND gates	XOR gates
AES	6,800	25,124
SHA256	90,825	42,029
MinCut	999,960	2,524,920

Table 2.2 Number of boolean gates

Garbling scheme	AES	SHA256	MinCut
GLNP15	0.59	3.41	69.04
Half-Gate	0.21	2.77	30.52
JustGarble	0.21	2.77	30.52

Table 2.3 Garbled circuit sizes in MB

2.4.1.2 Applications and Circuit Size

So, we have decided which protocol and garbling scheme. Now we can want to choose a function. The size and type of the resulting garbled circuits directly influence the efficiency of the protocol. For one, the Garbler must send the garbled circuit and its inputs to the Evaluator. The bigger the garbled circuit, the more data the garbled must send. The cost grows linear with the circuit size. Depending on the garbling scheme, the types of gates are also important. E.g., it might be better to have more XOR gates.

We selected three applications that have become the de-facto standard for benchmarking MPC protocols. They represent different orders of magnitude in circuit sizes as seen in Table 2.2. [83] uses AES to benchmark MPC protocols. Its circuit is tiny compared to others. MinCut, on the other hand, is a large circuit. It is the largest circuit available in libscapi. In addition, we chose SHA256 as it fits right in between AES and MinCut as a medium-sized circuit.

Another factor that influences the circuit size is the security assumption. Garbling with GLNP15 produces two ciphertexts for each AND gate and one for each XOR gate. However, using the Half-Gate construction or JustGarble outputs no ciphertexts for XOR gates and two for AND gates. We can see the effect of this in Table 2.3. The XOR gates used in MinCut have a strong influence on the circuit size for GLNP15.

2.4.2 Transport Layer

Using the provided parameters like circuit size, implementation, and input size, the transport layer of the framework will measure the packet loss and latency. All these factors determine which transport layer protocol the framework will choose. The framework will try to select the most efficient transport layer protocol for the given setup.

When choosing an optimal transport layer protocol, latency is a crucial factor. If the Round-Trip Time (RTT) is shorter than the transmission of one TCP windows, the sending buffer will fill. We want to keep our send buffer always filled for maximum performance. In low latency networks like LAN, congestion control is not that important. Usually, LANs do not suffer from packet losses. In such settings, the framework will choose UDP-based protocols.

Another crucial factor is the data volume and number of communication rounds. OT has small messages. Each message depends on a response, so it is not possible to send multiple messages at once. However, window-based protocols, such as TCP, are designed to transmit continuous data streams. The optimizations for large streams provide no benefit for protocols like OT. They negatively impact the performance due to the overhead and initialization phases. In these cases, UDP is a good fit since every message fits into a single UDP packet, and UDP has a low overhead. However, garbled circuits can be huge. So, choosing plain UDP is not a satisfactory solution. Even if we chose a different protocol like TCP, it would need to readjust to sending the circuits because it already adjusted for the small packets.

The final factor our framework uses for protocol selection is packet loss. When a packet loss occurs, plain UDT will fail, and TCP performs poorly, even with low packet losses of approximately 1%. In these situations, UDT is a fitting choice to ensure reliability. Packet loss is not only the effect of bad internet connections. Interferences in wireless networks (WLAN, 4G, 5G, ...) can also lead to packet losses.

2.4.2.1 Transport Protocols in our framework

We have integrated UDP, TCP, and UDT into our framework. On networks without packet losses, we used UDP as a reference. We chose UDT because of its increasing popularity and its impressive performance. It is TCP-friendly (i.e., it does not steal all the bandwidth for itself) and performs better than TCP in many situations. Once newer and more efficient protocols evolve, one can easily integrate them into our framework.

2.5 Implementation

The design goal of our framework is to provide an extendable modular design. It should be easy to extend it with other secure computation protocols as well as transport layer protocols. With the framework, Secure Computation researchers can focus on the protocol details without worrying about the networking aspects of the implementation.

2.5.1 Abstraction

The transport layer part is a wrapper. One only must know how to use the wrapper and not all underlying transport protocols. One can easily use it with secure computation protocols. Currently, it supports only synchronous sockets, which are sufficient for our purposes. The framework completely abstracts the network functionality. One can set the transport layer protocol at runtime. Changing the protocol at runtime removes the need for recompilations and makes it easier to compare different protocols. We used C++ polymorphism to achieve this modularity since most of the secure computation implementations developed in the past few years also use C++. The core of the abstraction is an abstract class with virtual methods. These virtual methods include functions like establishing and closing connections and sending and receiving data. Every transport layer protocol added to the framework must implement these methods. We require only four methods of the abstract class: `SetupClient`, `SetupServer`, `RecvRaw`, and `SendRaw`. This abstraction is very convenient for both developers of Secure Computation implementations and developers adding transport protocols. Adding new transport layer protocols does not require any knowledge about the Secure Computation application which will use it. Secure Computation developers link their applications against the provided shared or dynamic library. We can update the library independently so Secure Computation implementations can benefit from future features without recompiling. Our wrapper currently supports UDP, TCP, and UDT.

```
/* step 1 */
int one = 1;
struct sockaddr_in local;
int fd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
setsockopt(fd, SOL_SOCKET, SO_REUSEADDR,
           (const void *)&one, sizeof(one));
setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, (void *)&one,
           sizeof(one));

/* step 2 */
bzero((char *) &serveraddr, sizeof(serveraddr));
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
serveraddr.sin_port = htons((unsigned short) 1234);

bind(fd, (struct sockaddr *) &local, sizeof(local));
listen(fd, 1);
```

Listing 2.1 Example server setup in plain C/C++

2.5.2 Simplification

We predefined class methods to simplify tasks. The instantiation of the `Transport` class already allocates and sets up the socket (e.g., by using `socket()` for TCP or UDT. The application can then use this socket to create a server (aka listening connection) or a client. We can see how the number of lines significantly decreases from using plain C/C++ code, as shown in Listing 2.1, to using our wrapper, as shown in Listing 2.2. This reduction improves readability and encourages users to separate program logic from network code. Modularity and separation are important to make distinct parts reusable. It is also easier to evaluate individual parts.

To measure the latency between host and client, the wrapper also includes two static methods, `GetLatencyClient()` and `GetLatencyServer()`, which we can use to decide which protocol to use. These methods use UDP packets to measure the RTT in milliseconds.

```
auto *t = GetTransport("tcp");
t->SetupServer("0.0.0.0", 1234);
```

Listing 2.2 Example using our wrapper

2.5.3 Packet handling

Transport layers protocols usually send their data as packets or streams. UDP, e.g., sends packets for each request. Sending data up to the size of the Maximum Transmission Unit (MTU) will work fine. However, sending data that exceeds the MTU is impossible without splitting the data into smaller chunks. A program that wants to use UDP must implement these features. TCP, on the other hand, sends its data as a stream. It tries to fit as much data as possible in each packet and automatically splits data. Nagle's algorithm defines the behavior to wait for TCP packets to fill an IP packet [125] and can be turn off via the `TCP_NODELAY` option for faster sending of small data. The receiver must know what to receive because the sender glues together each payload with no indication where one ends and the other starts.

The wrapper abstracts these protocols and allows users to send and receive packets in arbitrary sizes. It defines a `Packet` class that contains the payload and the payload size. For UDP, it will automatically split the data, and for TCP, it will add size information, which acts as markers, to the stream. So, on the receiving side, we can always receive one packet. For the user, it does not matter if the packet got reassembled from multiple UDP packets or extracted from a TCP stream.

There are two ways to implement endianness: big-endian where the CPU would store the hexadecimal number `0xABCD` as `0xABCD` in memory, and the little-endian, the most common, where the CPU would store the same number as `0xCDAB`. When sending the data, the network payload will have this memory representation. Since the packet payload is just raw bytes without any type annotations, the receiver must know how it interprets the data. The wrapper converts all lengths to network byte order (big-endian) to ensure compatibility between machines with different endianness.

```
auto *t = GetTransport("tcp");
t->SetupClient("0.0.0.0", 1234);
t->Connect();
auto p = t->Recv();
t->Send(p);
```

Listing 2.3 Example of an echo client using our wrapper

2.5.4 Integration

Integration of the wrapper is easy. We can simply use the wrappers methods `Send()` or `Recv()` in the `Transport` instance. Supplying the protocol, we want to as a string to the class is optional. If one adds new protocols to the wrapper, applications using the wrapper can benefit from these protocols without rewriting. We can see a simple echo client in Listing 2.3. In the shown example code, a client connects to a server, receives data, and echoes it back to the server.

For our evaluations, we integrated the wrapper into libscapi. Libscapi uses external OT extension libraries, which bring their own (redundant) network code. We removed these portions and replaced them with calls to our wrapper. With these changes, it is possible to select a suitable transport layer protocol for garbled circuits and OT. Additionally, both can benefit from changes in the wrapper, and we removed redundant code. Our goal is to separate the network code and the application code and keep them self-contained.

2.6 Simulation

We demonstrate the effect of latency, packet loss, and bandwidth on the performance of Secure Computation protocols by performing simulations. These simulations provide an estimation for real-world network evaluations. We use `tc qdisk`. This command allows us to introduce latency and packet loss to network interfaces in Linux operating systems. For the host machine, we used a single Vultr instance with a 64-bit single-core CPU with 2.6 GHz and 2 GB RAM. Vultr provides simple to deploy VMs in various parts of the world. We used to loopback interface to perform all measurements.

Previous works only compared the performances between LAN and WAN. To get a more expressive image, we must examine the behavior to certain conditions in finer detail. Using simulation, we can fine-tune parameters and generate interim values.

We simulate latencies between 1ms to 300ms that represent latencies from LAN to various parts of the world. E.g., the RTT of a LAN is about 1–5ms, the RTT within North America is 50ms on average, and machines placed in North America and Asia, or EU and Australia are about 300ms. In addition to the latencies, we simulate packet losses between 0.01% and 0.05%. These packet losses represent packet losses in real networks [103]. For the bandwidths, we used 200Mbps, 500Mbps, 1Gbps, and 10Gbps in our simulation. We performed measurements using `iperf` and found these values to be most representative.

To understand the effect of latency on secure computation protocols, we use TCP, UDP, and UDT. The UDP results are the baseline for the best possible runtime. We optimized parameters (huge send/receive buffers) for fast and reliable networks. Of course, when encountering packet losses, UDP will fail since it does not provide any reliability features. As reliable protocols with use TCP and UDT. We show the bandwidth utilization for different circuit sizes. All experiments use only one thread/core and rely on hardware acceleration through AES-NI.

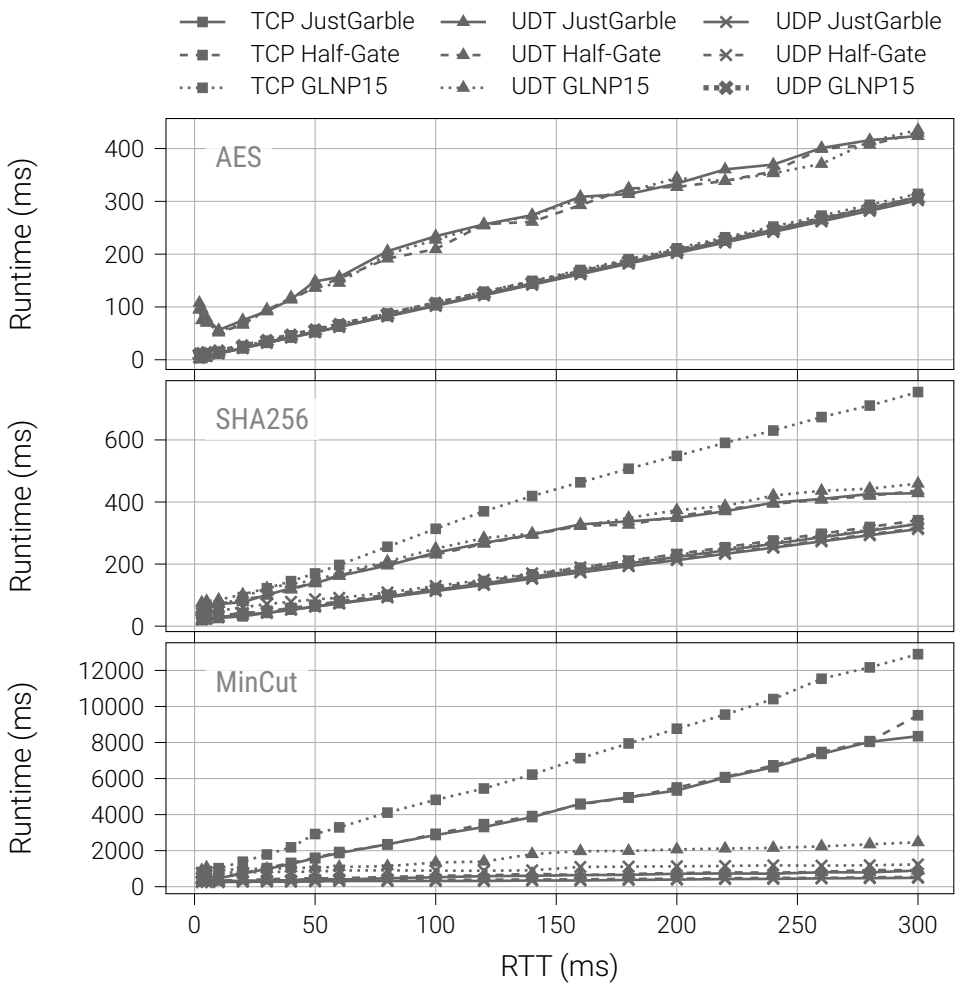


Figure 2.7 Effect of latency on a 10Gbps link for AES, SHA256, and MinCut

We can see in Figure 2.7 that for small circuits, such as AES, that TCP performs better than UDT when disabling Nagle's algorithm [125]. Small transmissions with only a few kilobytes cannot benefit from UDT optimizations. Gu and Grossman [82] optimized UDT for large data transfers. However, we can see that for the medium-sized circuits, such as SHA256, while TCP still performs better using Just Garble and Half-Gate protocols, UDT starts to outperform TCP for the GLNP15 protocol with increasing RTT. We can see that the performance of UDT is better as Bandwidth Delay Product (BDP) increases and when the amount of data transferred increases. For large circuits such as MinCut, we see that UDT utilizes the available bandwidth much better than TCP. The importance of congestions control mechanisms starts to show as the size of data sent from the Garbler to the Evaluator increases. We can

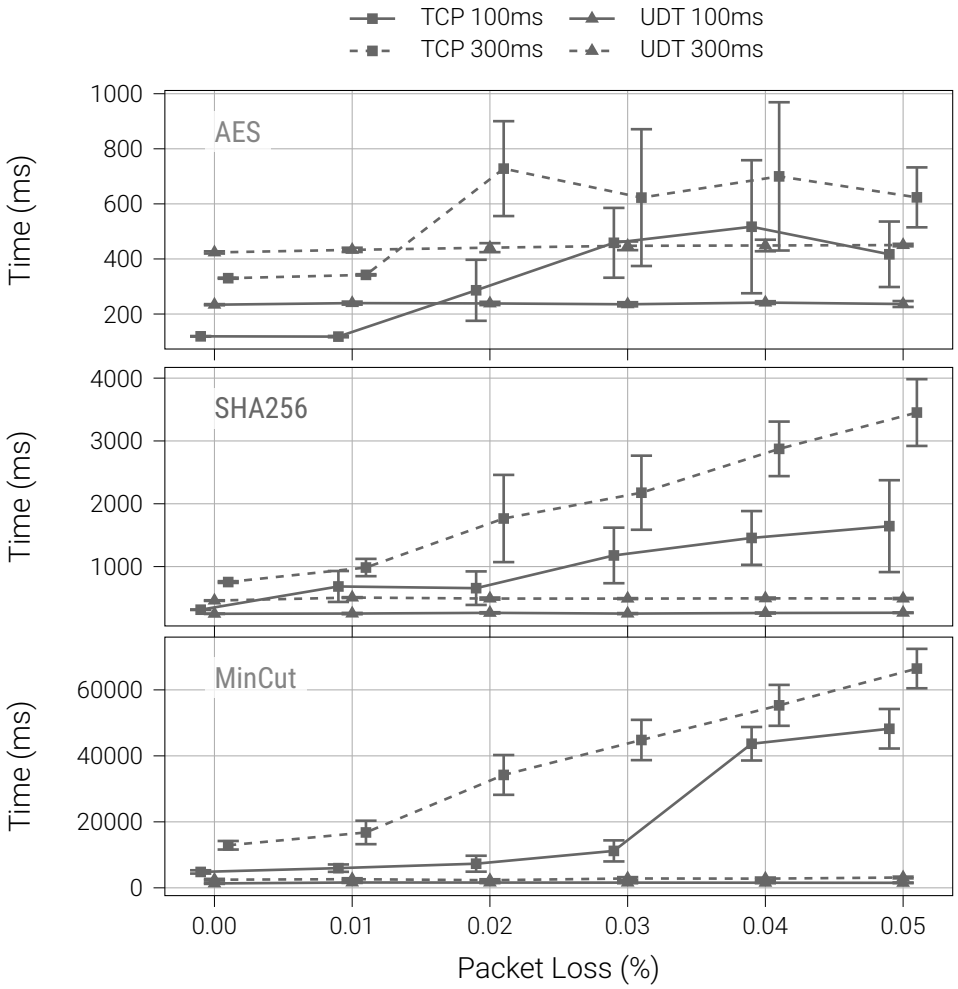


Figure 2.8 Effect of loss on a 10Gbps link for AES, SHA256, and MinCut

also see that higher latencies affect TCP than UDT as UDT does not react to packet loss/delay but uses a timer-based selective ACK.

We can see in Figure 2.8, that TCP starts to struggle even for small circuits, such as AES, when the packet-loss increases and UDT outperforms TCP. For medium and large Garbled Circuits, the performance of TCP decreases drastically. UDT manages packet loss much better with almost no visible impact.

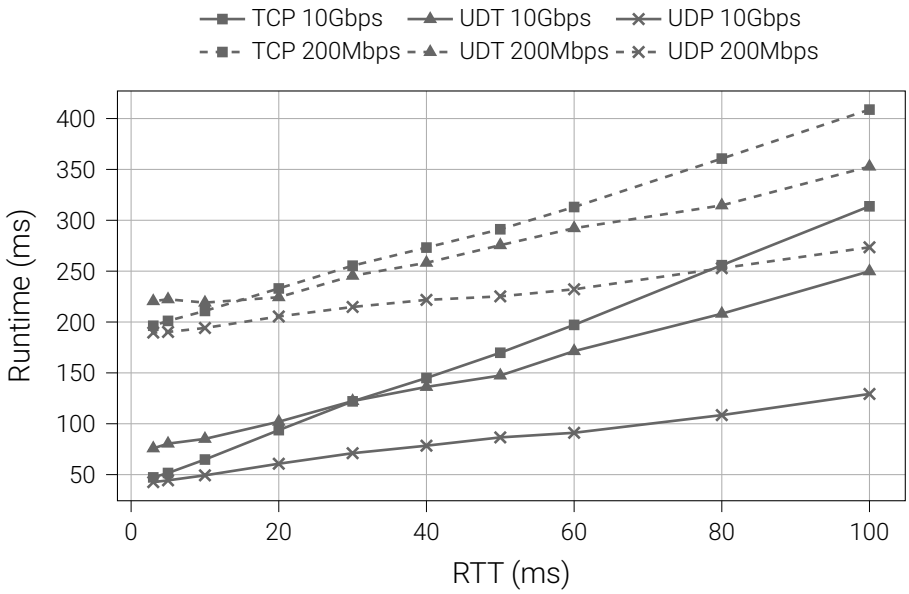


Figure 2.9 Performance of SHA256 using GLNP15

In Figure 2.9, UDT performs better than TCP at slightly higher latencies, above 15 ms, in 200 Mbps. When using high bandwidths like 10Gbps, UDT performs better when latencies are above 30ms.

Furthermore, we show the effect of bandwidth on the three applications we have considered. Figure 2.10 is the results for JustGarble protocol and Figure 2.11 is the results for GLNP15 protocol. We can see that the choice of protocol, i.e., JustGarble or GLNP15, impacts the performance for SHA256. When using JustGarble, TCP still performs better, but when using GLNP15, UDT performs better. We can also see that the delay affects all protocols equally.

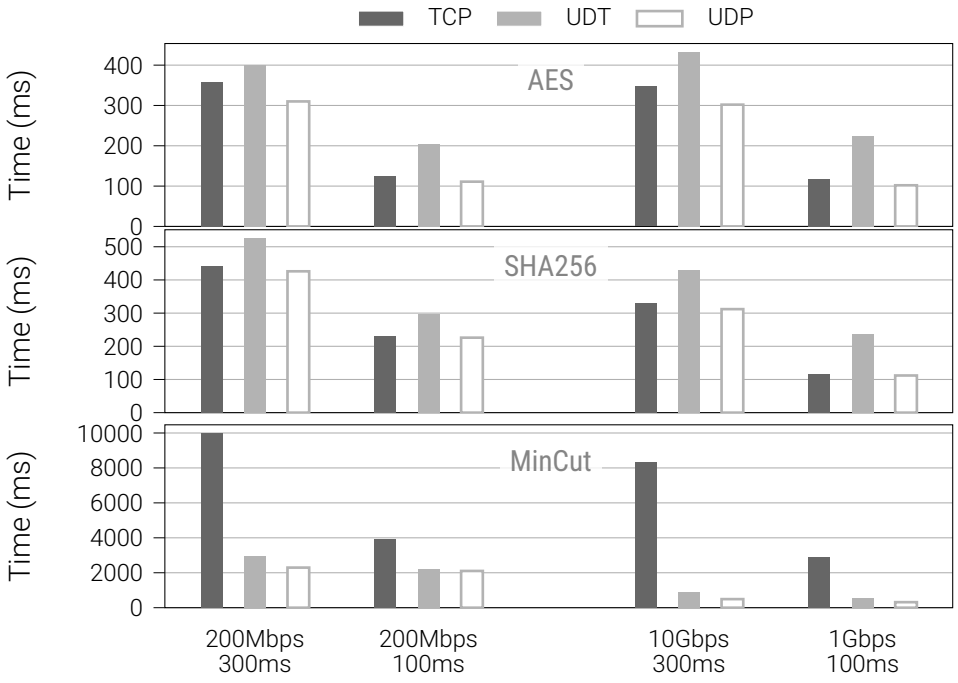


Figure 2.10 Performance of a JustGarble protocol

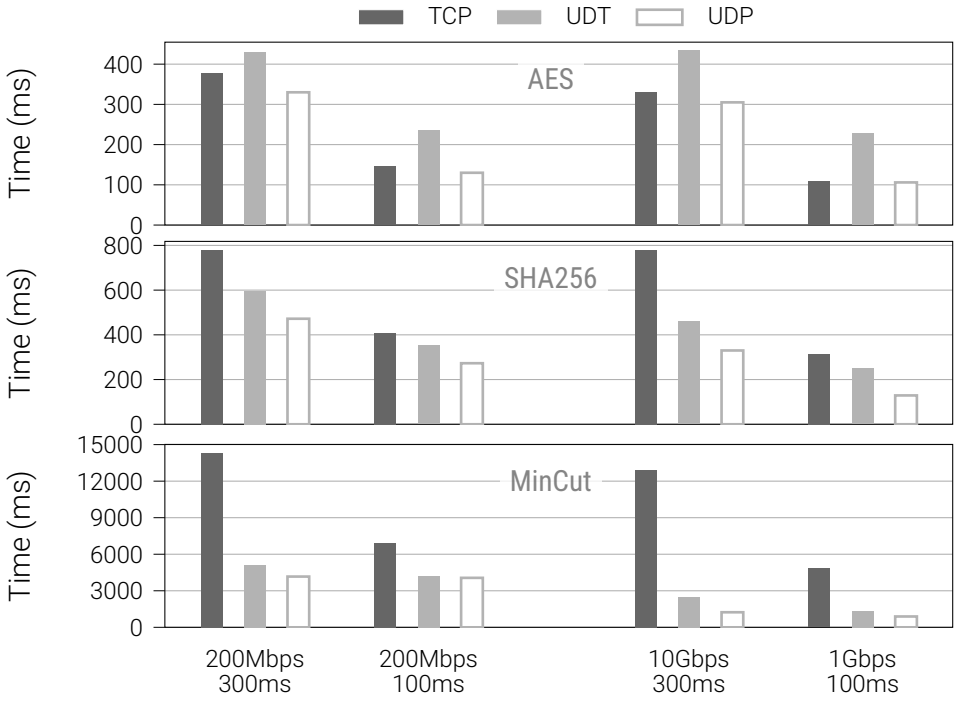


Figure 2.11 Performance of a GLNP15 protocol

2.7 Evaluation

We set up testbed environments to evaluate two different deployments: LAN and WAN. In both setups, we use two Azure VMs. The VMs had a 64-bit Intel Xeon quad-core CPU running at 2.4 GHz and 28 GB of RAM. However, all simulations only used a single core/thread. AES-NI support was available and used.

In the LAN setup, we use two Azure VM instances located in the same data center using a high-bandwidth network with low latency. The latency on a 10 Gbps link was 0.5 ms with a variance of 10%.

In the WAN setup, we used two pairs of locations with different latencies for our experiments. We can then observe the effect of latency and transport layer protocols on secure computation protocols on real networks.
For the EU-US setup, we located one VM in the EU and one in the central US. We estimated the network speed to be 1 Gbps with a measured latency of 110ms. The measured available bandwidth for a single TCP connection was 200 Mbps on average. Both machines run Ubuntu 16.04. The measured variance was 10%.
For the EU-AUS setup, we located one VM in the EU and one in southeast Australia. We estimated the network speed to be 1 Gbps with a measured latency of 300 ms. The measured available bandwidth for a single TCP connection was 100 Mbps on average. The measured variance was 20% for AES and SHA256, and 30% for MinCut using TCP, and 25% for MinCut using UDT.

Circuit	Setting	Just garble		Half-Gate		GLNP15	
		TCP	UDT	TCP	UDT	TCP	UDT
AES	LAN	**2.4**	187.1	**2.9**	191.3	**7.0**	202.7
	EU-US	**127.4**	403.9	**126.3**	408.3	**130.4**	444.2
	EU-AUS	**312.4**	566.8	**310.9**	580.2	**377.9**	592.5
SHA256	LAN	**13.5**	191.9	**19.9**	226.7	**30.5**	233.5
	EU-US	**146.2**	332.2	**152.0**	318.3	**266.5**	412.0
	EU-AUS	**362.5**	568.2	**394.1**	587.0	650.4	**612.4**
MinCut	LAN	**255.2**	598.2	**267.2**	740.2	**700.8**	1255.9
	EU-US	2616.6	**896.7**	2783.6	**957.6**	4911.9	**1802.3**
	EU-AUS	8204.6	**1069.1**	8693.6	**1163.1**	13805.2	**2001.3**

Table 2.4 Experimental results for garbled circuits protocols (runtime in ms)

We summarize the experimental results in Table 2.4. The results are an average of 100 runs. The table includes timings for garbling, data transfer, and output computation. TCP performs best for all circuit sizes in the LAN setting. One reason is that when disabling Nagle's algorithm [125], the kernel's network layer with send packets as soon as they arrive at the send buffer. Another reason is that the LAN setup is almost ideal in terms of bandwidth, latency, and packet loss. All optimizations UDT has are overhead because there are not degrading network conditions to compensate.

When running 2PC protocols in the EU-US WAN setting, circuit sizes impact the performance. For AES and SHA256, TCP is still more efficient than UDT. However, MinCut is 2.7–3 times faster by using UDT instead of TCP. Secure computation of MinCut using GLNP15 with UDT is more efficient than under JustGarble or Half-Gate using TCP.

When running 2PC protocols in the EU-AU WAN setting, the latency and packet loss strongly affect the performance. Secure Computation of SHA256 with GLNP15, with 223,679 ciphertexts, using UDT is slightly faster than TCP. For MinCut, it is 7–8 times faster than using TCP. This performance improvement exceeds the improvements one can expect from secure computation protocol improvements [166].

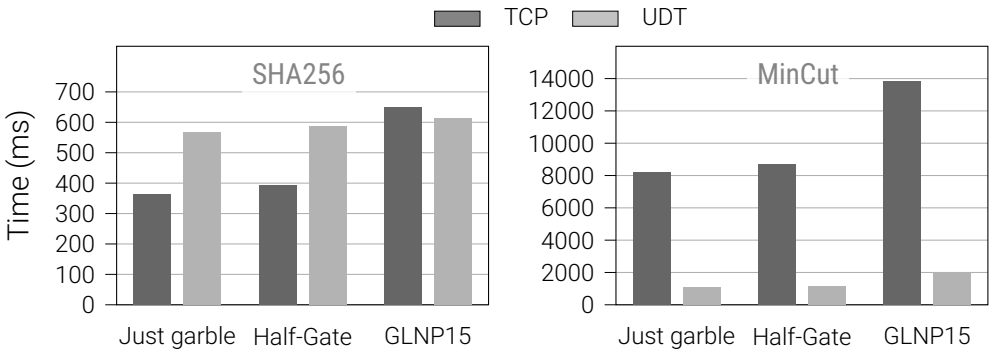


Figure 2.12 Comparison of assumptions between EU and AUS

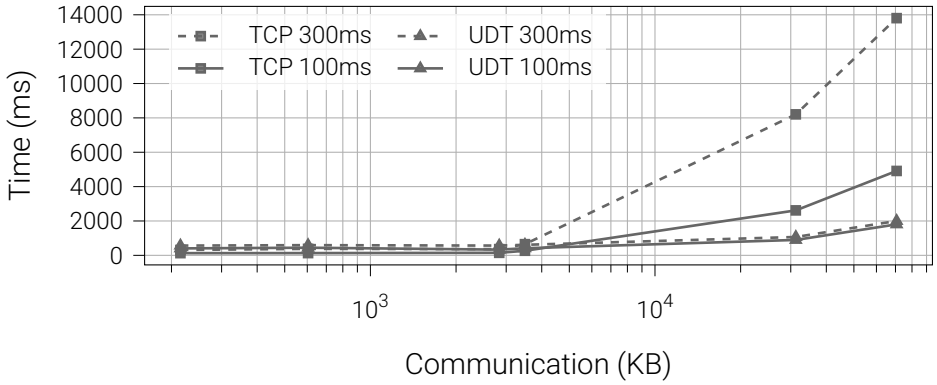


Figure 2.13 TCP vs UDT as garbled circuit size increases

Figure 2.12 shows a comparison of the performance of TCP and UDT under the three security assumptions. It further demonstrates the improvements which we can achieve by choosing an appropriate transport layer protocol.

We see a comparison of TCP and UDT for increasing ciphertext sizes in Figure 2.13. In this figure, we can see the impact of latency on TCP and UDT. The processing time for UDT increases by 200 ms when the latency increases by 200 ms. The processing time for TCP, however, significantly increases by magnitudes.

2.8 Conclusion

We demonstrate that bandwidth is not the performance bottleneck when choosing appropriate transport layers. We showed that we achieve performance improvements for 2PC implementations with general-purpose transport layers. E.g., UDT performed up to eight times faster compared to TCP. We also showed the benefit of modular transport layers in our proposed framework. This design allows one to easily change or extend transport layers without changing or recompiling the 2PC application.

We set up a testbed to perform our evaluation. Even if local evaluations provide a good benchmark, they do not reflect the behavior in real networks like the Internet. We showed the effect latency, bandwidth, and packet loss have in a simulation, comparing it to real network connections.

2.8.1 Future Research

We opened opportunities for follow-up research. One field might be to optimize transport layers specifically for certain 2PC implementations. These optimizations should adapt to parameters like communications rounds and the amount of data sent for each iteration. Also, they should consider the network conditions. Integration of such transport layer protocols will be straightforward using our modular design.

Another open field is the evaluation of different networks. While LAN and WAN networks are similar, some networks have different conditions. Other networks have become more crucial, like mobile networks (4G, 5G) or IoT networks (e.g., LoRaWan, Zigbee, Thread). These networks introduce different restrictions like packet loss and delay based on the position and limited payload sizes.

In our research, we only focused on 2PC and passive adversaries. A future using Multi-Party Computations or active adversaries is another possibility.

CHAPTER 3

Public Key Infrastructures

To use cryptography, we need encryption keys. Most platforms use asymmetric keys to reduce the number of keys. However, we must exchange these keys before we can use them. We use the term certificate to bind cryptographic keys to objects, like e.g., IP addresses, domain names, and email addresses. There are usually two forms to distribute these public keys and their corresponding certificates. In one, there is a consensus of multiple parties about which information we trust. The TOR network and PGP rely on this kind of key distribution. Another way is to define trust anchors. We create trust chains and assume that every chain that validates back to one of the trust anchors is secure. Of course, this is only true if we trust the trust anchors. But even if we trust these anchors, other problems arise. If we want to add a new entry to this Public Key Infrastructure (PKI), one party, which is part of a trusted chain, can add a new party. However, it will only do that if it can verify if that party owns the email address, domain name, or IP address, it claims to own. The parties must perform these verifications over the Internet, without any existing PKI to support this step. This step is troublesome and error-prone due to attacks. In this chapter, we will show the challenges of PKI validations. Since most services run using HTTP(S), we will base our examples on web servers and clients. Of course, one can expand this to everything that uses a PKI.

3.1 Introduction

The Public Key Infrastructure (PKI) is an essential part of the security of the Internet. It stores certificates that we can use to validate the identity and integrity of connections over the Internet. Certificates hold information about a public key and its corresponding Internet resource. To issue these certificates, we rely on Certificate Authorities (CAs). They own special certificates which allow them to sign other certificates. CAs are the trust anchors and the backbone of Internet security. Every valid certificate chain traces back to one CA. If a client requests a certificate for a domain, like example.org, the CA will verify if that client owns that domain and then issue a certificate. The certificate contains numerous fields, including the domain name and the public key of the server. The CA also adds a signature to the certificate using the content of the certificate and its private key. When a client connects to the server using the domain example.org, the server sends its certificate, and the client can verify it. On success, the client will start to request data. Otherwise, clients usually report an error to the user.

To perform these validations, Browsers include hundreds of registered CAs. We consider every certificate successfully tracing back to one of these CAs trusted. Correctly verifying the ownership of the domains is crucial since one corrupted CA can put the whole PKI in jeopardy. There are multiple approaches to validate the domain ownership: Domain Validation (DV), Organization Validation (OV), and Extended Validation (EV). DV is the simplest. It provides multiple mechanisms, like sending emails to that domain or checking for challenges and responses in the domain records. All these mechanisms work automatically without human interaction from the CA. DV assumes that only the owner of a domain can receive emails for specific addresses or modify the domain's records.

Although OV and EV provide much higher security by requiring human interactions, it is inefficient for the same reason. The process involves steps like communication with the customer, phones calls, requesting additional documents, ... This manual verification also takes longer, making instant certificates like in DV impossible. Also, it is very costly, and prices can exceed $1000. Of course, one can also trick humans doing the validation by, e.g., social engineering, but our research focuses on DV since it is the most used mechanism nowadays. DV is fast, automated, cheap, and supported by 99% of all CAs.

After a CA issues a certificate, it should publish it to be accessible to all users. Lightweight Directory Access Protocol (LDAP) directories are a common choice for that. For the verification, this is not necessary. E.g., web servers usually send the whole certificate chain, excluding the root certificate, to clients, so they do not need to fetch any additional information. However, CAs must provide information about the revocation status of a certificate. It must be possible to withdraw a certificate even if the certificate has not expired yet. E.g., a reason to revoke a certificate may be a leaked private key. Certificate Revocation List (CRL) is the current way to maintain revocation information. It contains a list of all revoked certificates. Clients can identify the certificate as revoked if the CRL includes it. Another option is to use the Online Certificate Status Protocol (OCSP). With OCSP clients can query a server for a certificate. Usually, CAs operate their OCSP servers. When querying a server for a certificate, the server will reply with the status good, revoked, or unknown.

3.1.1 Contribution

We show how to manipulate domain validation mechanisms and obtain certificates. We exploited the ownership verification with an off-path attack using DNS cache poisoning. Using this attack, an attacker can pass domain validations and receive fraudulent certificates for domains it does not own. We successfully launched the attack against 7 CAs that use domain validation. However, even a single vulnerable CA is enough to subvert PKI security. The PKI's security falls with its weakest link, and a domain signed by one root CA is valid for every Internet browser.

Before our work, such off-path attacks were considered theoretical. Attackers actively use DNS cache poisoning but use MitM techniques [63,143]. We demonstrate DNS cache poisoning by using off-path attacks.

We developed a BGP simulator to evaluate BGP paths on the Internet. Our simulator achieves higher performance and accuracy than other simulators. In combination with our simulator, we analyze the resilience of the domains ecosystem to attacks against domain validation. Our measurements show that the domains ecosystem is not resilient to prefix hijacks. It reveals that only a few ASes own most of the domains and that many have all their name servers on a single AS.

We discuss mitigations. Cryptographic protection of DNS would prevent the attacks, but full deployment is far from being complete. Thus, we propose the distributed domain validation as a drop-in replacement for the standard domain validation. It allows strong resistance against MitM attackers. We also show that many IPs are anycast. While anycast does not provide any security for the standard domain validation, distributed domain validation benefits from anycast IPs. We also analyze the validations agents' placement on the Internet and propose ASNs for hosting these. We also demonstrate a method to determine ASes that are good for agent placement.

3.2 Background

We will define some basic background information needed for the following sections. These include the domain validation itself and its underlying components.

3.2.1 Organization of Internet resources

The Internet Assigned Numbers Authority (IANA) is the central organization that supervises Internet resources like IP address blocks, the DNS root zone, Autonomous System Numbers, and more. IANA does not manage all these resources and delegates them to one of the five Regional Internet Registries (RIR) that manage resources for a specific region. The first is Réseaux IP Européens Network Coordination Centre (RIPE NCC), founded in 1992. It operates in Europe, Central Asia, Russia, and West Asia. The Asia-Pacific Network Information Centre (APNIC), founded in 1993, provides Internet resources to East Asia, Oceania, South Asia, and Southeast Asia. The American Registry for Internet Numbers (ARIN), founded in 1997, serves Antarctica, Canada, parts of the Caribbean, and the United States. The Latin America and Caribbean Network Information Centre (LACNIC), founded in 1999, is responsible for the Caribbean (excluding the parts from ARIN) and Latin America. Finally, the African Network Information Center (AFRINIC), founded in 2004, serves Africa. These RIRs allocate address blocks to Local Internet Registries (LIRs). Most of the LIRs are Internet Service Providers, enterprises, and academic organizations. If a, e.g., a small company in the United States needs an IP address block, it would get this from an LIR that is a member of ARIN. Membership is a requirement for every LIR. That small company would then receive an IP block from that LIR. This IP block is part of ARIN allocated IP blocks.

```
$ORIGIN example.org.
$TTL 86400
@       SOA dns1.example.org. hostmaster.example.org. (
            1970010101 ; serial
            21600      ; refresh after 6 hours
            3600       ; retry after 1 hour
            604800     ; expire after 1 week
            86400 )    ; minimum TTL of 1 day

        NS          dns1.example.org.
        NS          dns2.example.org.
        MX      10  mail.example.org.

dns1    A           192.0.2.1
dns2    A           192.51.100.1

mail    A           192.0.2.3

host    A           192.0.2.4

ftp     CNAME       host.example.org.
www     CNAME       host.example.org.
```

Listing 3.1 Example zone file for example.org

3.2.2 Domain Name System

The Domain Name System (DNS) is a hierarchical system for name lookups. Without DNS we could only access computers on the Internet by knowing their IP addresses. With DNS we can use names. While one can also use it in private networks, when talking about DNS without any further explanation, we are talking about the DNS of the Internet. A DNS zone contains all information about a domain and its subdomains. We usually refer to DNS zones as zone files since they are often (but not always) plain text files containing all information. We can see an example of such a zone file in Listing 3.1. We call each entry in a zone a record. The most important record is the SOA (Start of Authority) record. It contains information about the contact email (in this example hostmaster@example.org) and the primary name server. Name servers answer queries for DNS records. The name server in the SOA answers queries for this domain. The serial, refresh, retry, and expire information is for secondary name servers, which pull from this name server. With this information, secondary name

Hostname	IPv4	IPv6	Operator
a.root-servers.net	198.41.0.4	2001:503:ba3e::2:30	Verisign, Inc.
b.root-servers.net	199.9.14.201	2001:500:200::b	University of Southern California
c.root-servers.net	192.33.4.12	2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13	2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10	2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241	2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53	2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17	2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30	2001:503:c27::2:30	Verisign, Inc.
k.root-servers.net	193.0.14.129	2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42	2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33	2001:dc3::35	WIDE Project

Table 3.1 All 13 DNS root servers

servers know how often to check for updates and when to stop. A different serial number indicates a change in the zone for the secondary servers. The TTL (Time to Live) field is for negative caching, i.e., how long a secondary server should cache failed queries.

Besides the SOA, we can also see an NS (name server) record, an `A` (IPv4 address) record, an `AAAA` (IPv6 address record, pronounced quad A), and an `MX` (mail exchange) record. There are also two `CNAME` (Canonical Name) records that functions as an alias. These records point from one domain to another. There are other records, but the most important DNS records for our research are `NS`, `A`, and `CNAME`. In Listing 3.1, we can also see an `$ORIGIN` and a `$TTL`. The first defines the origin of the zone file, in this case, `example.org.`. The name server will append to everything ending without a full stop the domain `example.org.`, e.g., `dns1` will become `dns1.example.org.`. The TTL, however, is particularly important for our research. It defines how long a DNS resolver should cache the response.

DNS resolutions are recursive. There are thirteen root servers on the Internet. These globally distributed servers allow redundancy and avoid single points of failure. We can see a list of all root servers in Table 3.1. For every recursive lookup, we start from one of these root servers. Since there is no way to lookup root servers, these

servers are hardcoded into every DNS resolver. As an example, we want to look up the IP address for `example.org.`. We assume we have no information cached and must perform a full recursive resolution. The full stop at the end indicates a root server. We query one of the root servers for a name server that answers queries for `.org`. The first domain, i.e., the last part of a domain name like .com, .net, ..., is the Top-Level Domain (TLD). In our example, the name server replies with a randomized list of DNS name servers for `.org`. The randomization is for load-balancing reasons, and we pick the first one, e.g., d0.org.afilias-nst.org. Only if a query to that server fails, we fall back to the second entry. We ask the current name server for the name server of `example.org.`. This server then replies with the name server for that domain, which is in this example `a.iana-servers.net.`. Now we have to do the same steps again and query the root servers for `.net`. Then query, e.g., `e.gtld-servers.net.` for `iana-servers.net.`. This server then outputs a list with all name servers for that domain and the corresponding IP addresses. Now we can query `a.iana-servers.net.`, since we know the IP address, for the A record of `example.com.`. The response includes the IP address 93.184.216.34 and a TTL of 86400 seconds (1 day). Because the resolution involves so many steps, DNS resolvers cache the result and answer future queries using the cached value until the cache expires. So, when a DNS resolver recursively resolves a domain, it will also cache all intermediate steps. In addition to the query, a name server may supply information the client did not request. E.g., when asking for a name server, it may already return the IP addresses of all name servers. We call these glue records. They speed up the lookups and keep the number of queries needed small. However, sometimes this information is crucial to avoid infinite loops, where for instance we require `.net` to resolve `.org` but the `.net` name server relies on another `.org` name server.

Writing a recursive DNS resolver is a non-trivial task. Because of the high demand of all Internet users and potential loops, developers must rely on glue records and other caching strategies. These optimizations can also lead to security flaws like cache poisoning, which we will explain next.

3.2.3 DNS cache poisoning

With DNS cache poisoning, an attacker manipulates the resolver's DNS cache to return incorrect data. If we modify a DNS resolver's cache to point `example.org` to an IP we control, it will answer to succeeding queries with our IP address, effectively hijacking that domain and impersonate it. We can achieve this in multiple ways. One way would be to intercept the connection between the resolver and the name server by, e.g., a Man in the Middle (MitM) attack. We will also show a way to achieve this using an Off-Path attacker later in this chapter. When receiving a DNS response, the resolver checks if the Transaction ID (TXID) and source port are the same. According to RFC5452 [102], clients should use random ports and TXIDs. Both fields are 16 Bit. Initially, DNS resolvers only used the TXID to validate responses. After the announcement of Kaminsky's cache poisoning attack, DNS resolvers received patches lowering the initial $1/2^{16}$ down to $1/2^{32}$. In addition, RFC6056 provides recommendations for TCP port randomization. Another way is to abuse glue records. Depending on the implementation of the DNS server, it might cache glue records we provide for another domain.

Attackers can use DNS cache to distribute malware, steal credentials, enforce censorship [10], or surveillance [99]. Financial gain can be another use case where attackers use this technique to steal money or cryptocurrencies.

The Internet Engineering Task Force (IETF) designed and standardized in RFC4033 to RFC4035 [13–15] the Domain Name System Security Extensions (DNSSEC) to mitigate DNS cache poisoning attacks. However, DNSSEC requires significant changes to the DNS infrastructure and the protocol. The initial proposed standard dates to 1997 [67]. Sadly, DNSSEC is still not widely deployed, and most countries did not pass 50% yet [11].

3.2.4 WHOIS

The WHOIS protocol allows querying databases that store information about Internet resources like domain names, IP address blocks (sometimes called prefixes), and Autonomous Systems. RFC3912 [60] defined the current version of the protocol in 2004. Every RIR provides a database for the resources they delegated. E.g., we can use RIPE's database to look up the registrant of an IP block. For the resources, there are different blocks like administrative or technical contact or information about the LIR. If we look up Google's Public DNS Server using a whois lookup, we see that ARIN delegated the prefix 8.0.0.0/9 to Level 3. Level-3 then delegated the prefix 8.8.8.0/24, which is part of that block, to Google.

3.2.5 Domain Validation

Domain Validation is by far the most popular choice for ownership validation. We extracted 122 Root CAs from Windows (Internet Explorer), macOS, iOS, and Linux (Mozilla). Out of these 122 CAs, 51% support DV. These 51 CAs control more than 99% of the certificate market share [128,162].

The process to obtain a certificate using DV is in most cases as follows. An applicant generates a Certificate Signing Request (CSR) using OpenSSL or a similar tool. The CSR contains information such as the organization name, the domain name, the country, and the public key. The applicant uploads this CSR to a CA including other information, like which DV method to use. The CA will then use the supplied DV procedure to validate the domain.

There are multiple methods for DV and not all CAs support all of them. We show a list of which CA supports which methods in Table 3.2. All methods rely on DNS which makes them vulnerable to DNS cache poisoning. The DV methods are as follows:

Email-Based DV When an applicant selects this option for DV, the CA will send an email. Of course, the applicant cannot freely decide which email to use. Instead, the CA provides a list of emails to choose from, like, e.g., admin@example.org. The email contains a validation code that the applicant can enter on the CAs website or a link to complete the validation. If we provide a valid code by entering it on the CAs web page or opening a link from an email the link, the CA will issue a certificate.

	Email	DNS	HTTP(S)	WHOIS
COMODO	✓	✓	✓	
DigiCert				✓
Entrust				✓
GeoTrust	✓	✓	✓	
GlobalSign	✓	✓	✓	
GoDaddy	✓	✓	✓	
Network Solutions	✓			
SSL.com	✓	✓	✓	
SwissSign	✓			
Thawte	✓			
Trustwave				✓
Symantex				✓
Startcom	✓			
Let's Encrypt		✓	✓	
Unizeto	✓	✓		
NETLOCK	✓		✓	
IdenTrust				✓
RapidSSL	✓	✓	✓	
StartSSL	✓			
Certum	✓	✓	✓	
InstantSSL	✓	✓	✓	

Table 3.2 List of supported DV methods of CAs

WHOIS-Based DV This method is similar to the Email-Based DV. The main difference is that the applicant cannot choose the email address at all. The CAs will select an email address from the Admin, Registrant, Tech, or Zone contact of the domains WHOIS record.

DNS-Based DV For this DV method, the CA requests the user to modify the zone of the domain. The CA will calculate a challenge and a response for the applicant to set up. These challenges and responses are usually cryptographic hashes. E.g., we assume that an applicant wants to get a certificate for the domain example.org. The applicant must generate a CNAME record with the challenge that points to the response:

```
<challenge>.example.org.    CNAME    <response>.example.org
```

The CA's DNS resolver then queries the name server. It checks if the CNAME record with the challenge is present. If it also points to the correct response, the CA will issue a certificate.

HTTP/S-Based DV This method is like the DNS-Based DV method. However, instead of modifying the DNS records, the applicant must place a file on a webserver. E.g., the file name is the challenge, and the content is the response. So, if an applicant wants to verify the domain example.org, the CA will make an HTTP(S) request for a file on a predefined directory, e.g., the root directory or `.well-known`, named `<challenge>.txt` and check that the content of the file equals the response. If `https://example.org/<challenge>.txt` is valid, the CA will issue a certificate.

3.3 Related Work

Since our work combines different research fields, we must cover a broader range of related work. In this section, we will show work related to our research arranged into distinct categories.

3.3.1 DNS cache poisoning

We already knew about the potential dangers of cache poisoning in the late nineties. Even so, researchers warned about the vulnerability Dan Kaminsky was the first to demonstrate an attack in 2008 [107]. The attack targets DNS resolvers that used fixed or incrementing source ports. Following the publication, RFC5452 [102] recommended patches, including randomizing source ports or randomly selecting the name servers. However, these patches did not last long and were, shortly after their introduction, circumvented. [94,97] demonstrated how to predict ports using side channels. [95,96,144] bypassed the patches by using fragmentation. However, these attacks do not guarantee a successful attack. If the DNS resolver already has cached a value, it will ignore DNS records until the cache expires. [111,148] studied the behavior of DNS resolver caches to find out in which conditions resolvers will overwrite the cached value. Furthermore, in [110,140], researchers analyzed multiple actors of publicly shared DNS resolution platforms and the caches of these platforms. [145] examined the behavior of various forwarders in DNS resolution chains that lead to cache poisoning vulnerabilities.

In theory, DNSSEC [13–15] prevents cache poisoning attacks. However, we see vulnerabilities and misconfigurations in DNSSEC key generation and management. These flaws affect more than 35% of all signed domains [53,58,146]. [90] showed that DNSSEC could not prevent replay attacks that redirect clients to incorrect servers in Content Distribution Networks (CDNs).

3.3.2 CA Compromises

A web browser assumes that a connection is secure if it can validate the certificate chain, and that chain ends with one CA certificate on its list. Issuing a certification and validating ownership requires no human action. These properties spawned a big certificate market. However, more CAs also mean more dangers. One vulnerable or compromised CA is enough to subvert any domain.

There is a long history of attacks against CAs. A bug in Debian's OpenSSL from 2006 to 2008 resulted in about 26k vulnerable certificates [71]. The flaw resulted in private keys having only 15-17 bits of entropy. In 2011, an attacker compromised DigiNotar [30]. Vendors immediately had to remove that CA from their products. Security proposals included directions for users to manually remove DigiNotar from their list until the vendors patched it. Attackers signed a wildcard domain (valid for all subdomains) for google.com to spy on Iranians. In 2015, attackers obtained a certificate for Microsoft's live.fi domain. It is part of Microsoft's Live email server, and attackers registered the email hostmaster@live.fi, which Microsoft did not block, and no other user already used. Using Email-Based Domain Validation, the attacker successfully proved their ownership to Comodo. In 2015, Egyptian ISP MCS holdings received an unconstrained intermediate certificate from China Internet Network Information Center (CNNIC). MCS used to certificate for deep packet inspections of encrypted traffic in firewalls. However, this certificate could sign any domain and MCS. With this certificate, CNNIC violated root CA policies, resulting in the removal from, e.g., Mozilla's Root CA list. In 2016, the Chinese CA Wosign issued certificates for GitHub due to a faulty domain validation [141]. If a user could prove ownership of a subdomain, the CA would assume ownership of the base domain. [9, 70, 98] documented loopholes in PKI ecosystems. In addition, many browsers accept 1024-bit RSA keys. We can assume that nation-state attackers can break keys with that bit size [20].

[28,29] also pointed that how attackers can use the Border Gateway Protocol (BGP) using prefix hijacking to bypass domain validation.

3.3.3 PKI defenses

TLS and the PKI received new security mechanisms to prevent attacks or limit the consequences. These include Certificate Transparency (CT), HTTP Strict Transport Security (HSTS), HTTP Public Key Pinning (HPKP), and TLS fallback Signaling Cipher Suite Value (SCSV), which prevents downgrade attacks. However, there are also attempts to create alternative PKIs and proposals for supplementary entities to store and check certificates. One attempt is monitoring CAs' behaviors. Examples for this are CT [117], Sovereign Keys [72], Accountable Key Infrastructure (AKI) [109], Attack-Resilient Public Key Infrastructure (ARPKI) [21], and distributed PKIs [4, 48, 151, 159]. The DNS-Based Authentication of Named Entities (DANE) [69] relies on DNSSEC [13–15]. It provides a list of trusted CAs that can issue certificates for that domain. Another approach that lists acceptable CAs is the DNS Certificate Authority Authentication (CAA) Resource Record [89]. These methods complement existing certificates. Their biggest downsides are the overhead and complexity involved, which complicates deployment and thus sees no adoption. Since January 2015, the Chrome Browser utilized CT and, since June 2016, started to display undisclosed EV certificates as untrusted [81].

In 2004 [131] proposed CoDNS, intending to improve the availability and performance of DNS lookups. In CoDNS, trusting nodes agree to resolve each other's queries when their lookups are failing. The adoption of CoDNS requires modifications of the DNS resolvers. Our proposed solution works with no further changes to the existing servers. The main goal of CoDNS is to improve performance and not to improve security. A single malicious DNS resolver can compromise the whole DNS lookups.

ConfiDNS [134] extends CoDNS. It uses peer agreements to ensure security to queries. It forwards DNS responses from multiple hosts to a local resolver and uses per-site lookup histories. However, the collection of DNS responses centrally without further security mechanisms creates another attack surface.

Perspectives [159] builds a notary system where hosts can lookup known keys. Notary hosts keep track of server keys. Clients can query the log and check if the key matches.

DoubleCheck [7] prevents attacks when retrieving certificates for the first time. It uses a remote host over TOR to use an alternative path. This solution, however, is unsuitable for domain validation since it assumes correct certificates.

After making our DV++ implementation available in March 2017, [28,29] and LetsEncrypt proposed a similar approach, called multi-VA. They locate these VAs in the same ASes (ASN16509 - Amazon, ASN13649 - Flexential Colorado Corp.). In contrast, we ensure that nodes are not in the same AS. Also, we check that the paths between the nodes and name servers do not overlap.

3.4 Off path attacks against Domain Validation

In this section, we show how we can maliciously obtain a certificate for a domain. We trick the CA into issuing a certificate, which we do not own, by utilizing DNS cache poisoning with off-path attacks. We inject our mail server into the DNS cache to successfully path the Domain Validation (DV) process. The attack combines different components. First, we trigger a DNS request by the CA to our name server. Then, we must match the challenge-response values with a specially crafted spoofed DNS response. We will describe all the steps necessary and present the attack. Further, we will provide a measurement study of the CAs and the potential victim domains. This study provides an estimate of how many clients and servers are vulnerable to our attack.

3.4.1 Triggering the DNS Request

The first obstacle we must overcome is triggering the DNS request. The DNS resolvers from CAs do not answer external queries. However, we can use the CA's website and force a DNS query. First, we upload a CSR. Then, the CA extracts the target domain from the CSR and provides us with all supported DV methods and their options. As our target domain, we choose example.org.

3.4.2 Defragmentation Cache Poisoning

Our next obstacle is to generate a spoofed DNS response, including our name server, which the CA's resolver will accept. Our period for the attack begins with the initial DNS request and ends with a timeout or the correct response. For a valid response, we need the matching source port and TXID. According to RFC5452 [102], DNS resolvers should randomize these values.

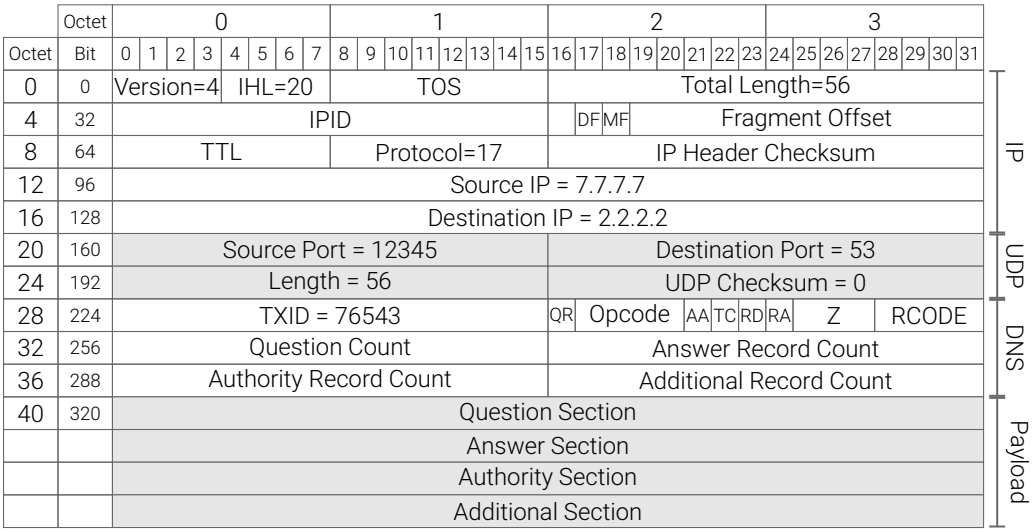


Figure 3.1 DNS request packet

Figure 3.1 shows the structure of a DNS request. In this example, we see a DNS request from the DNS resolver at the IP address 7.7.7.7 to a name server at the IP address 2.2.2.2. The resolver sends the packets from port 12345 to port 53 of the name server. The TXID is 76543. With the source ports and TXID fields being 16 bits, we have 32 bits of entropy with 2^{32} values. An attacker must spoof a DNS response with the correct port and TXID, which seems like an impossible task. However, we will show how to circumvent these restrictions and successfully poison a DNS cache using off-path attacks using overlapping IPv4 fragments. We use these fragments to manipulate legitimate DNS responses. We overwrite the DNS answer but keep the original port and TXID. We will show how to force a server to use fragmentation by limiting the Maximum Transmission Unit (MTU).

3.4.2.1 Forcing IP fragmentation

The MTU is the largest size of bytes that one can transmit over a single network transaction [136]. The Path MTU from two endpoints is equal to the smallest MTU on the path. E.g., we cannot send packets bigger than the MTU. To circumvent this limitation, the Internet Protocol includes fragmentation. With fragmentation, we can split big packets into smaller ones. Most networks support a MTU of 1492 bytes (with PPPoE overhead) or 1500 bytes [122]. RFC791 [136] defines the smallest possible MTU to 68 bytes. Since most DNS responses do not exceed 1500 bytes, they do not get fragmented. Of course, there are exceptions like DNSSEC, which exceeds 1500 bytes due to its many cryptographic records.

For our attack, we must force a low MTU, such that the first fragment contains the source port and TXID, which are too hard to guess, and the second fragment contains the information we want to overwrite.

Path MTU discovery [124] is a mechanism to probe the lowest MTU on the path. Hosts set the Do Not Fragment (DF) bit in the IP header. When we enable the DF bit, routers along the connection do not fragment the packets. Instead, they will reply with an error including Type 3 (ICMP Destination Unreachable) and Code 4 (Fragmentation Needed and DF set). On receiving such an error, the Operating System will store this information. Linux, e.g., will store this information for 10 minutes. Everyone on the Internet can send such an ICMP error message. Since the sender does not know all the routers on the path, it will accept ICMP error messages from every IP address. Thus, we can use these messages as an off-path attacker to force a host to assume a smaller MTU. We measured how many servers reduce the MTU using this technique. We used the Alexa Top 5K list, where 33,4% up to 296 bytes. 11% even reduced it below 296 bytes. 80% of the servers will reduce the packet size to 600 bytes or lower. The minimal value in the Linux kernel is 552 bytes by default, while RFC1191 [124] recommends 576 bytes.

Unlike ICMP with TCP headers, Operating Systems, e.g., Linux 3.13, do not verify ICMP messages with UDP headers since UDP is stateless. Thus, we can easily spoof ICMP Fragmentation Needed errors.

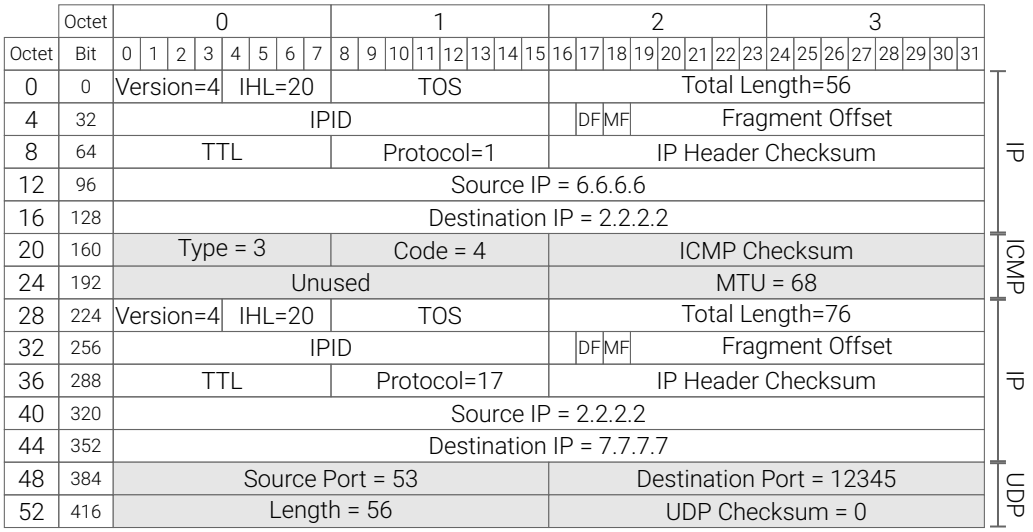


Figure 3.2 ICMP fragmentation indicating an MTU of 68 bytes

In Figure 3.2, we see an example ICMP Fragmentation Needed error sent from attacker IP 6.6.6.6 to a name server at IP 2.2.2.2. The payload of the ICMP packet is the IP header and the first eight bytes of the original packet that caused the error. The error tells the name server at 2.2.2.2 to reduce its MTU to 68 bytes when connecting to the DNS resolver at 7.7.7.7.

3.4.2.2 IPv4 Fragmentation Reassembly

When a host receives fragments of an IP packet, it stores them in a defragmentation cache, by default for 30 seconds. It will use the IP ID value to identify which IP fragments belong to the original IP packet. All IP fragments of a single IP packet will have the same IP ID. The offset field indicates the order. However, IP fragments can overlap, i.e., the offset of an IP fragment is smaller than the size of the previous fragment. The latest IP fragment will overwrite data from earlier IP fragments. The More Fragments (MF) bit indicates that the current packet is incomplete, and more IP fragments follow. The last IP fragment will have this bit set to zero.

In our attack, we exploit IP fragmentation. We trick a DNS resolver into reassembling the first fragment of a valid response with our second fragment, bypassing the security mechanisms of DNS. However, we still must predict the correct IP ID. The IP ID must be unique for every source-destination pair according to the RFC791 [136]. Operating Systems usually use one of three algorithms [77]. One is to increment the

value, either globally or per destination. Another way is to use random IP IDs. From the Alexa Top 10k domains, 60% used globally incrementing values, 40% incremented values per destination, and less than one percent used random IP IDs.

While Windows uses a globally incrementing IP ID, Linux Operating Systems default to increments per destination. Attacking the globally incrementing IP ID is straightforward. An attacker probes its victim using IP connections and measures the increase rate. The attacker can then determine the IP ID the name server will use to send its DNS response. But even if the server increments the IP ID per destination, we can still estimate the IP ID. For our attack, we used the method presented in [112]. The algorithm is complex and not part of this work, and we refer to [112] for a detailed explanation. The last option is to use fully randomized IP IDs. It introduces overhead because the server must check for collision, which we do not need for incrementing IDs. For that reason, not many servers use randomized IP IDs. The IP ID field is 16 bits. A randomized IP ID would be the hardest out of the three with a success probability of $1/2^{16}$. For all three attacks, we can raise the success probability by sending multiple fragments. If we could send 2^{16} fragments, our success probability would be 100% for randomized IDs. However, Operating Systems limit the number of IP fragments it stores, e.g., Windows only allows 100 fragments, and recent versions of Linux limit this to 64 IP fragments.

Another obstacle is the UDP checksum. RFC768 [135] describes the calculation of the checksum. It is a simple algorithm designed for error correction and not for security. Receivers of UDP datagrams will discard them if the checksum is incorrect. When reassembling the fragments, the UDP checksum must be valid. So,, if we send our second fragment, the assembled datagram must still have the same checksum. Because the algorithm is so simple, we can efficiently achieve this. We force a fragmented response to our IP address. Then we modify the second fragment and calculate its difference. Remember, the content of the first fragment changes every time. However, if we maintain the same checksum for our modified second IP fragment, the checksum of the reassembled packet will be the same. We can change the checksum to any value by appending two bytes. Hosts will strip these remaining bytes because they exceed the value from the UDP length field.

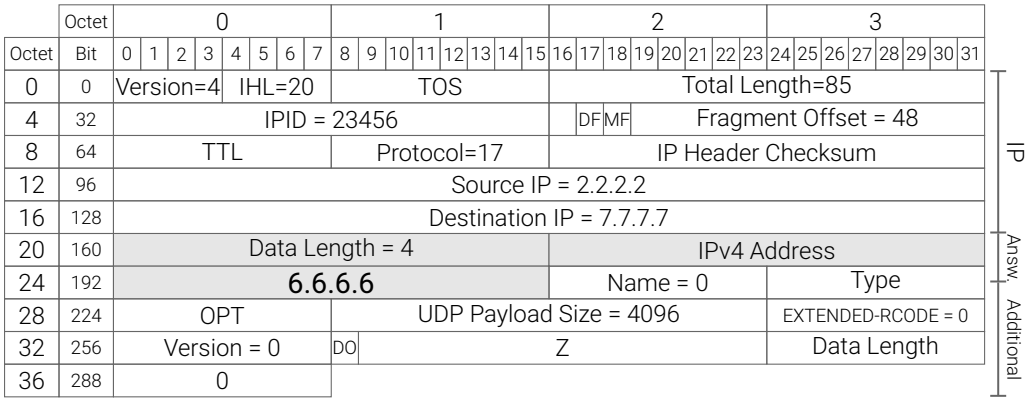


Figure 3.3 Malicious second fragment modifying the mail server to the IP 6.6.6.6

3.4.2.3 Exploiting fragmentation

Let us assume we want to get a certificate for the domain vict.im. First, we must do some preparations. We need to collect a fragmented DNS response and determine which fragmentation we need using the ICMP Fragmentation Needed error. Once we know the MTU we will use, we modify the second fragment and fix the checksum by appending bytes. Our next step is then to estimate the IP ID. Once we have these requirements, we can place our second fragment in the cache of the CA's DNS resolver. This fragment changes the mail server to point to our IP address. Figure 3.3 illustrates an example of such a second fragment. We must ensure that the DNS resolver and the name server use the MTU we previously determined using the ICMP Fragmentation Needed error. In our example, we use the E-Mail Based Domain Validation. The CA cannot send a verification code to, e.g., hostmaster@vict.im if it does not know the IP address of the mail server responsible for that domain. Therefore, the CA must resolve the mail server (MX) and its IP address (A). Our attack lowered the MTU, and the CA will receive fragments. In our example, we have chosen an MTU of 68 bytes. For the IP resolution, the CA's DNS resolver will receive two IP fragments. The first fragment contains the correct port and TXID. We show an example of the first fragment in Figure 3.4. However, our second fragment is already in the cache. The DNS resolver will discard the original second fragment considering it as duplicate. Now that the CA seems to know the mail server, it will send a mail. However, since we modified the second fragment to contain our mail server, we will receive the mail with the validation code. Now, we can finish the Domain Validation by entering the validation code on the CA's website. With the finished Domain Validation, we can download our signed certificate.

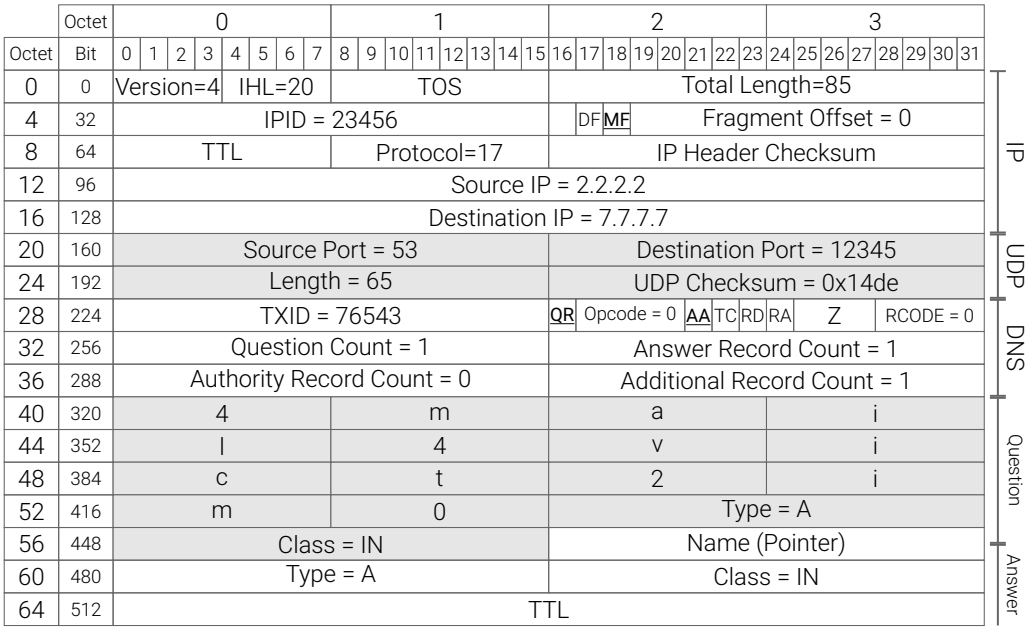


Figure 3.4 First fragment from the name server 2.2.2.2 to the resolver at 7.7.7.7

3.4.3 Overwriting DNS caches

The attacks we presented allow us to bypass DNS's security mechanisms. However, there is still one catch. The method only succeeds when the domain is not in the DNS resolver's cache. Very often, the DNS cache already contains popular internet domains. In that case, the DNS resolver will ignore the information from the response and use its cache. The next step is to trick the resolvers and overwrite their caches.

There is no general approach to force DNS resolvers into overwriting their caches. Different DNS implementations have individual logic. RFC2181 [74] recommends ranking priorities, i.e., which values have precedence over others. The implementation, however, differs from software to software. We can use these differences to fingerprint the software. We characterized DNS caches of CAs and evaluated under which conditions new values can replace the cached ones. The study builds upon the evaluations of [111].

3.4.3.1 Setup

We used the domain vict.im and its subdomains for our study. The zone file contains six name servers. Each name server had a corresponding IPv4 (A) and IPv6 (AAAA) address, three mail servers (MX), and other records like anti-spam records (SPF). We cause CAs to send DNS requests, which we monitor. For the response, we built a tool using Standford's DNS server library in Perl. Finally, we use the collected information to categorize the DNS caches and their overwriting behavior. We selected payloads presented in [111]. We measured the minimum MTU values of popular Alexa servers and found that often we can reduce its value to 68 bytes.

3.4.3.2 Vulnerable Certificate Authorities

We found that some CAs share the same infrastructure. We will group these CAs. We also found that the following CAs were vulnerable to overwriting attacks:

COMODO, InstantSSL, NetworkSolutions, SSL.com These CAs use the same mail server (mcmail1.mcr.colo.comodo.net) and the same caching DNS resolver. Our study indicates that the resolvers use Bind 9 with DNSSEC validation.

Thawte, GeoTrust, RapidSSL These also share a single mail server and the same DNS resolver.

StartCom, StartSSL StartCom stopped issuing new certificates in January 2018. Both use the same mail server and the same DNS resolver.

SwissSign Uses BIND 9.

3.4.4 Challenges

The attack does not work against all CAs. We require the following conditions: The CA must use an automated Domain Validation method. The DNS resolver and the network of the CA must allow fragmented responses. Also, the domain's name server, which we attack, must not filter ICMP fragmentation needed error messages. If the resolver's cache is not empty, the resolver must be vulnerable to at least one of the overwriting attacks.

Another big challenge is combining all attacks to trick the domain validation. Since off-path attackers cannot monitor any communication of its victim, the attacker must predict events. Triggering the query, measuring the IP ID, and constructing the spoofed second fragment must be conducted in the correct order and timing. We can only indirectly trigger DNS queries by using the submission from the CA's website. This condition makes it is harder for us to fingerprint DNS resolvers and to analyze their caching behaviors.

Additionally, one must conduct some preprocessing before the attack. Some servers do not fragment at the forced position. E.g., if we try to reduce the MTU to 512 bytes, the server might use 500-byte fragments. The attacker must adjust the attack by using padding or utilizing compression [123] in DNS to make it work.

Another hurdle is when servers use randomized IP IDs. In our research, we used a name server with a globally increasing IP ID. Depending on the load on a name server, estimating IDs can be quite easy or quite challenging. It may take multiple attempts for the attack. Fortunately, name servers are not as crowded as, e.g., web servers. It even gets worse if name servers use randomized IP IDs. For randomized IDs, the attacker must predict the pseudo-random sequence. Another option is to try the attack multiple times using brute force. However, random IP IDs are rare on the Internet.

3.4.5 Mitigations

We have shown that it is feasible to attack domain validation using off-path cache poisoning attacks. We identified a few vulnerable CAs. However, it does not matter how many CAs are vulnerable since only a single CA is sufficient to issue a certificate for every domain.

3.4.5.1 Blocking fragments

Blocking all fragments seems like simple mitigation. Without fragmentation, there would be no fragmentation cache poisoning. However, MTUs smaller than 1500 bytes are natural on the Internet. E.g., most DSL users have an MTU of 1492 bytes because Point-to-Point Protocol over Ethernet (PPPoE) requires 8 bytes.

Using data captures from the Center for Applied Internet Data Analysis (CAIDA) we examined how common fragmentation is on the Internet. CAIDA runs two Internet monitors in the Equinix datacenters in Chicago and San Jose. These monitors capture packets and anonymize those. The resulting dataset is available as a gzipped PCAP file. We analyzed 14484 traces in 121 datasets, representing data from 9 years (2008 - 2016). We observed that one in 135k packets in an ICMP fragmentation needed packet. Our measurement reveals that blocking traffic would block many Internet clients.

Also, just blocking fragments would not fix the vulnerability. Attackers may use other approaches to bypass randomization or poison DNS resolver caches. Off-path attackers are the weakest attackers. It is not uncommon that attackers launch MitM attacks, e.g., using BGP prefix hijacks [142].

3.4.5.2 DNSSEC

The IETF standardized the Domain Name System Security Extensions (DNSSEC) in RFC4033 to RFC4035 [13–15] to mitigate DNS cache poisoning attacks. DNSSEC even prevents MitM attackers. DNSSEC extends DNS with cryptographic functions, which allow owners to sign their domains and prevent modification. DNSSEC prevents DNS cache poisoning when fully deployed. However, two decades after DNSSEC proposal and standardization, only 27% of DNS resolvers validate DNSSEC [11], and only 1% of all domains are signed [163]. [146] and [53] show problems with DNSSEC key generation. It is unclear when these issues get resolved and when we will see a fully deployed DNSSEC. However, without these requirements, we cannot see DNSSEC as mitigation yet.

3.5 Optimized BGP Simulation for Evaluations

While BGP simulators are not essential for the PKI or domain validation, it is crucial for our Internet evaluations. The Internet is a distributed interconnection of multiple networks. We cannot take parts offline or attack parts of the Internet without damaging critical infrastructures. Also, by just using the Internet, we know nothing about the routing of packets. Not all routing policies are public. Thus, simulating BGP paths is a suitable alternative.

We developed a simulator for the evaluation of BGP prefix hijack attacks as well as the evaluation of countermeasures against hijacks. Our simulator uses a dataset of target networks, e.g., hosting Internet domains, and the CAIDA AS relationship, i.e., customer to provider or peer to peer dataset [47], to determine the paths between ASes. CAIDA analyzes data traces and infers relationships from these. The relationship dataset has specific routing rules, i.e., zero or multiple customer-to-provider connections, zero or one peer-to-peer link, and zero or multiple provider-to-customer connections.

Our simulator uses breadth-first search taking into consideration also the AS relationships. Other simulators, e.g., [55, 78], ignore the relationships and generate a simple non-directed graph. The result is low performance and inaccuracies in measurements. The goal of those simulators is statistical evaluation, e.g., the success probabilities of attacks, rather than resolving individual paths. There are also very advanced simulators like GNS3 [147] that can simulate networks by fully simulating each router. However, doing this for over 65000 nodes is not practical.

Additionally, we optimized memory allocations and implemented a new feature: finding the BGP paths. We do not need advanced features like simulating BGP message exchanges between routers. Instead, we focus on performance and provide simulations within minutes in contrast to hours with other simulators. We also implemented non-optimized simulators using different algorithms to cross-validate the results.

# of ASes	71431
Average degree	11.07
Average mean neighbour degree	1054.41
Highest ASN	399383

Table 3.3 ASN statistics using CAIDAs dataset from March 1st, 2021

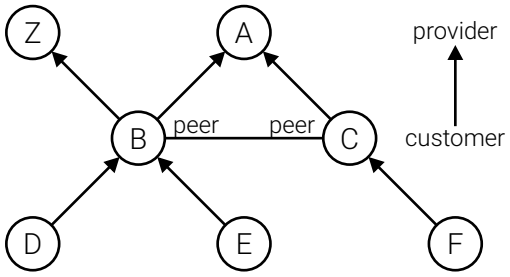


Figure 3.5 AS cashflow: Customers pay their providers for traffic

3.5.1 Implementation

We wrote our simulator in C++ and highly optimized it for speed. It preallocates memory and structures used for the search and thus avoids dynamic memory allocations during simulations. The simulator first parses the graph then we use the resulting data to perform the search.

3.5.2 Correct interpretation of relationships

The simulator reads Caida AS relationship data [47]. It supports both versions, Serial1 and Serial2. The relations between ASes follow the so-called cash flow. Table 3.3 shows the statistics of a recent dataset. Many stub ASes have just one neighbor, their uplink ISP. The row with the average number of neighbors shows this. That is why we included the average number of neighbors the neighbors have. In Figure 3.5, we see an example illustrating the cash flow. The arrows indicate where the cash flows, i.e., customers that pay their providers for traffic. ISP B and ISP C are peering ASes. They connect their customers to the customers of the other AS (usually without requiring any payments). The rules for a correct path are simple: start with zero or more customer-to-provider links, followed by at most one peer-to-peer connection, finally followed by zero or more provider-to-customer links. Figure 3.6 shows examples for valid and invalid paths. The paths (a) and (b), however, (b) would be the path of choice because it is shorter and traffic between ISP B and ISP C is free. Example

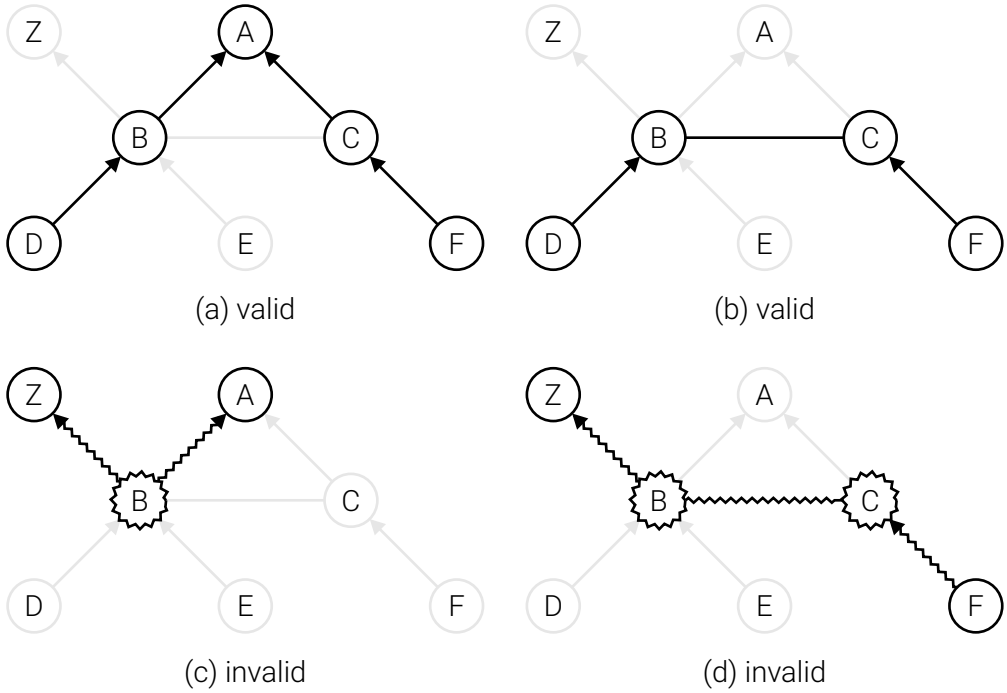


Figure 3.6 Valid and invalid paths

(c) is invalid since ISP B would have to pay for the traffic its uplink providers Z and A caused. Similarly, in (d), ISP B must pay for the customers of ISP C. ISP C would have to connect to ISP Z directly and not through ISP B.

Following the CAIDA relationship rules of the datasets has two advantages. For one, the results are more accurate. Just feeding the dataset into a graph and performing a breadth-first search would find invalid paths. And the main advantage is the performance boost. For a breadth-first search, we append all neighbors to a queue. When we pop a node from that queue, we add its neighbors. If we keep out invalid nodes, we keep this exponentially growing queue smaller. The result is a crucial performance improvement.

Offset (ASN)	Value
0	0
1	0
2	5
3	2
4	0
5	1
	...

(a)

Offset (ASN)	Value
0	NULL
1	Pointer to AS1
2	Pointer to AS2
3	Pointer to AS3
4	NULL
5	Pointer to AS5
	...

(b)

Table 3.4 Performance optimizations using direct memory lookups

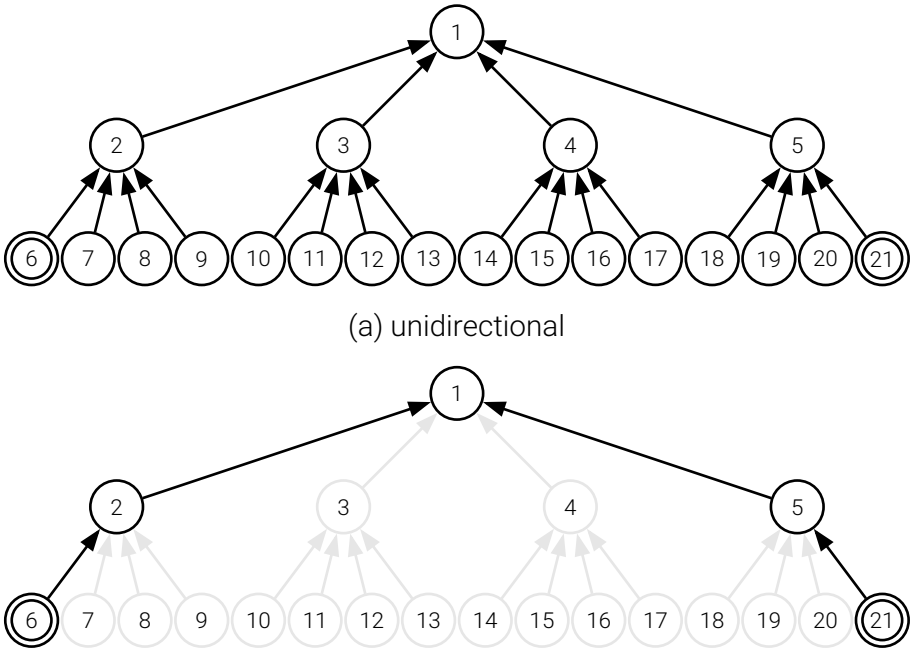
3.5.3 Fast lookups

The search algorithm can only traverse nodes once. Otherwise, infinite loops occur. To check if we visited a node already, we use a static array of bytes where the index is the Autonomous System Number (ASN). We can see an example in Table 3.4 (a). The first column is the ASN. In the second column, we store if we traversed the node. A zero indicates we have not traversed this node yet. But instead of adding one to the node we visit, we add the node that led to this visited node. By doing so, we can backtrack the table to retrieve the path. In the simplified example, we want to find the path from 1 to 3. Once we reach 3, we can backtrack to 2, 5, and 1, so we know the path is 1 -> 5 -> 2 -> 3.

Usually, we would create a dictionary datatype to get back the object using the ASN as an integer. But we can speed up AS lookups the same way we did with the visited nodes. We initialize a static array with the size of the highest ASN. We can see in Table 3.4 (b) an example. Of course, we waste space because there are gaps in the ASNs. However, for a single pointer, we need 8 bytes on a 64-bit machine. With 399,384 array items, we waste 3,195,072 bytes.

3.5.4 Bidirectional Search

We performed a test run calculating the path for one AS to 67306 ASes. It took 4 hours and 6 minutes to complete, which was still too slow for our use case. As a further improvement, we developed a bidirectional search algorithm. Providers



(a) unidirectional

(b) bidirectional

Figure 3.7 Node expansion for the path search from 6 to 21 using unidirectional search and bidirectional search

usually have multiple customers. So going down from a provider to its customers will add many customers to the queue. And since these customers may also have customers, the queue size grows exponentially. We can see the result of that in Figure 3.7 (a). To get from 6 to 21, we must add all nodes to the queue. First, we add the only neighbor 2. Successively, we add all customers and providers of 2. When we reach 1, we add all its customers and its customers' customers. Since our example is symmetrical, it does not matter where we start. On asymmetric graphs, like the Internet, it can make a difference from which node we start.

Providers have many customers, but customers have only a few providers. Using the rules of the relationship dataset, we take advantage of this property to optimize the search by using a bidirectional approach. We avoid adding customers. If a path exists between the source and the target, they must meet at a certain point when we search from two directions. In Figure 3.7 (b) we can see the benefit of that. Starting from 6 and 21, we only add providers and finally meet at node 1. This smaller queue provides a significant performance improvement. The test run with 67306 paths that took 4 hours and 6 minutes now only takes 15 minutes.

The performance optimization using a bidirectional search is also crucial for the future. The Internet is getting more shallow which means that the average path length is reducing. These shorter paths cause problems for search algorithms. Nodes have more neighbor nodes, and with each hop, the number of nodes to traverse grows exponentially. There are many stub ASes, so the average number of neighbors is 10.7. However, 1.3% of the nodes have more than 100 neighbors, and 0.16% have more than 1000 neighbors. Because these are so well connected, it is highly likely to traverse them while searching, and so the queue gets filled up with all the neighbors. Without our optimization, searching the graph becomes increasingly difficult.

3.6 Distributed Domain Validation

In this section, we present Distributed Domain Validation (DDV). We want to provide a design that protects against BGP prefix hijacks [18] and DNS cache poisoning [95]. Domain Validation must not be possible for anyone that does not control the domain. Also, an attack should not prevent a legitimate owner of a domain from passing Domain Validation. We achieve this by performing multiple lookups from various points on the Internet. Our DDV approach avoids overlapping paths on the Internet. It will only accept the majority results, ignoring lookups with different answers that might be malicious.

We designed DDV to be easy to adapt. Using it does not require any changes to existing name servers, and CAs can easily integrate it. CAs can trigger the DDV by using an API call. The API performs the DDV process and returns the result.

Bandwidth requirements for DDV are insignificant. It can run on the weakest AWS instance (t2 nano) with 512MB and a single shared CPU core and still have plenty of unused resources left. Even network requirements are small and can be satisfied with a 1 Mbit connection. In terms of bandwidth, 1GBs of traffic should yield at least 100.000 DNS lookups. However, it is a requirement that one host the agent using different ISPs, ideally in various geographic locations. Otherwise, an attacker can compromise the validation by attacking a single ISP. The more ISPs used, the better. The chances of successfully attacking multiple ISPs at the time are almost impossible. Our evaluations show that by using the top 10 ISPs or ten independent cloud providers, we could securely, i.e., at least two non-overlapping routes to the target, validate 80% of all domains in the Alexa, Cisco, and Majestic top 1 million list. These lists also contain misconfigured domain names, e.g., have only one name server. Both software components, namely the orchestrator and the agents, run on Linux distribution, OpenBSD, FreeBSD, and even Windows and macOS.

3.6.1 Design and Implementation

DDV is a decentralized system. It consists of two elements: an orchestrator and agents. The orchestrator coordinates its agents to perform domain validation. Each agent is equally trusted, and the result depends on multiple agents. The number of the correct responses needed from the agents is a parameter that one can change. The agents must be in separate locations and ISPs. For a successful attack, the attacker must compromise at least 50% of the agents. The orchestrator and the agents use HTTPS for their communication.

3.6.1.1 Agents

The agents are lightweight HTTPS servers set up on virtual machines on multiple cloud platforms. Each agent also uses a hardened Unbound DNS resolver. The orchestrator connects to the agents. The agents can perform the validation or return specific records, e.g., the target domains' name server. Each agent has an X.509 certificate known to the orchestrator. The certificate uses the IP address of the agent as the subject name, which is crucial since we do not want to rely on another PKI. We encrypt the communication between the orchestrator and the agents. However, the DNS queries issued by the agents to the name servers are unencrypted. A vast majority of the name servers on the Internet do not support encryption. The agents perform caching of records in Top Level Domains (TLDs) but do not cache hierarchies under the TLDs.

3.6.1.2 Orchestrator

The orchestrator coordinates the validation. However, it relies on the agents to decide. The agents and the orchestrator communicate over a secure channel, using the X.509 certificates of the agents. The orchestrator has a directory that contains all trusted certificates. It will not rely on any root servers. Only certificates in that directory are valid. We want to prevent any chicken-and-egg problems by keeping our component independent from existing PKIs.

3.6.1.3 Distributed Domain Validation

Once the orchestrator receives the responses from the agents, it compares their responses and selects a majority vote, which it then returns to the calling API. Hosting the agents and taking the majority answer is already an improvement for security. However, we propose a BGP-aware method.

Plain DDV

With the plain DDV, we place the agents across the Internet using different ISPs and locations. The orchestrator will use the majority answer. But what happens if multiple connections from agents go through the same malicious router? To prevent this scenario, we check the BGP paths in the BGP-aware DDV method.

BGP-aware DDV

In the BGP-aware method, the agents return the result and their BGP path. We use our simulator the calculate the paths between the agent and the target name server. In addition to the answers, the orchestrator uses the BGP path to avoid agents with overlapping BGP paths.

Given a domain to look up, the agents perform the following steps. They check if the name servers for the requested TLDs (e.g., .com) are in their cache. If that is the case, it uses the cached records to look up the name servers for the root domain (e.g., example.com). If the TLD is not in the cache, it queries the root name servers. Each root server uses anycast for load balancing, which also improves security against attackers. If the agents' responses do not match, the orchestrator will continue to query other agents with non-overlapping BGP paths. We ran an evaluation to see how many independent paths exist for popular domain names. We found that for most domains (>80%), there are at least two separate paths. This evaluation also includes misconfigured domain names that contain only one name server (either via an NS record or an SOA record). If the DNS response of agents returned matches, the validation succeeded, and the orchestrator will return the result.

3.6.2 Evaluations

We performed different evaluations to understand the DNS infrastructure and evaluate the feasibility. These include TTL measurements for caching of root server responses, a comparison of latency and failures comparing DDV to standard domain validation, and an evaluation of name server distribution between different countries and RIRs. We examine path lengths and analyze anycast IPs. We also analyze prefix sizes and their effect on security. Furthermore, we simulate attacks against DDV and calculate the best ASes for agent placement. All measurements use a combined list of the Alexa, Cisco, and Majestic top 1 million list.

3.6.2.1 Top-Level Domain TTL measurements

Since every recursion starts at the root servers, it makes sense to reduce the load on the root servers and speed up the validation by caching values. The root servers do not change very often since they rarely add new TLDs. We analyzed the time-to-live (TTL) values of top-level domains (TLDs). We took a list of 1508 top-level domains from `.aaa` to `.zw` and performed name server lookups for these on all 13 name servers. Additionally, we performed this test from multiple locations. All but one TLD return 172800 seconds (2 days) as their TTL value. Only `.arpa` returns 518400 seconds (6 days) as its TTL value. These results validate our initial assumption. We can use this information to cache the name servers for all top-level domains safely, reducing the load on the root name servers and speeding up the domain lookups. However, we do not cache name servers except for the TLDs.

3.6.2.2 Latency and failures

We set up non-caching recursive resolvers at various locations. We want to measure the latency and failure rate of DNS validation. For the baseline, we used our ISP just running the recursive resolver. We will call this the local setup. We also set up multiple agents at nine geographically diverse locations: Amsterdam, London, New York, Paris, Seattle, Singapore, Sydney, Tokyo, and Toronto. We will call this the distributed setup. For each agent, we add the round-trip time from the orchestrator to the agent. We take the first two responses we receive. The time measured for the distributed setup is when the second response from an agent arrives. If we could not resolve a domain, e.g., misconfigured name server, or if the request timed out, we count this as a failure.

	failed (domains)	failed (%)
local	28265	28.27%
distributed	21	0.02%

Table 3.5 Local vs. distributed resolution errors

	min	mean	max
local	5.23 ms	214.71 ms	4806.68 ms
distributed	16.32 ms	375.98 ms	3609.02 ms

Table 3.6 Local vs. distributed resolution latency

We took Alexa's Top 1 million list and truncated it to the top 100.000 to speed up the simulation. Table 3.5 shows how many resolutions failed. Due to geographical distribution, the agents resolve more domains without errors since the orchestrator can already stop when a majority exists. However, the distribution comes with round-trip times from the orchestrator to the agents, as shown in Table 3.6. We can see the additional overhead of the connection between the orchestrator and the agent. However, on average, the resolution using the distributed setup takes only 75% longer than the local setup. But to fully reflect the advantages of the distributed setup, we must consider that distributed setup could resolve domains that failed on the local setup.

Alexa		Cisco		Majestic		Combined	
US	34.30%	US	50.71%	US	36.88%	US	38.55%
DE	8.19%	DE	5.29%	DE	7.06%	DE	7.24%
TR	5.07%	FR	3.38%	FR	5.04%	FR	4.55%
RU	4.60%	CN	3.35%	GB	3.77%	RU	3.51%
FR	4.52%	GB	3.20%	JP	3.75%	GB	3.38%
IR	4.13%	CA	2.85%	RU	3.60%	CA	3.18%
GB	3.04%	RU	2.78%	CA	3.28%	NL	2.80%
CA	2.95%	NL	2.43%	CN	3.10%	TR	2.76%
NL	2.51%	JP	1.37%	NL	2.92%	JP	2.75%
BG	2.27%	BR	1.36%	BG	2.21%	CN	2.64%

Table 3.7 Name server IP distribution by country

	Alexa	Cisco	Majestic	Combined
afrinic	0.96%	0.57%	0.80%	0.78%
apnic	12.96%	11.62%	14.19%	13.13%
arin	37.14%	53.53%	40.12%	41.64%
lacnic	2.32%	3.09%	3.04%	2.92%
ripencc	46.62%	31.19%	41.86%	41.52%

Table 3.8 Name server IP distribution by RIR

3.6.2.3 Name server distribution

In Table 3.7 and Table 3.8, we see the distribution of all name servers IP addresses by country and by regional internet registries (RIRs). Usually, RIRs are responsible for specific regions like RIPE NCC (Réseaux IP Européens Network Coordination Centre) is responsible for Europe, West Asia, and the former USSR. However, Autonomous Systems can get prefixes from other RIRs. E.g., perl.com has five name servers. One of these name servers is ns3.us.bitnames.com, which resolves to 136.144.52.122. The 136.144.52.0/23 prefix for this IP address belongs to Equinix Services, a US company. The AS number (AS54825) belongs to the range that IANA assigned to ARIN (American Registry for Internet Numbers). The prefix belongs to RIPE NCC. A whois lookup at ARIN will redirect to whois.ripe.net, showing RIPE NCC as a maintainer in the "mnt-by" field. As a result, Table 3.8 is not just a summary of Table 3.7. We can see that the US/ARIN and Germany/RIPE have the most name servers located in their

ASN	Name	#IPs
46606	UNIFIEDLAYER-AS-1 - Unified Layer, US	4.04%
16276	OVH, FR	4.04%
16509	AMAZON-02 - Amazon.com, Inc., US	3.55%
24940	HETZNER-AS, DE	3.54%
32475	SINGLEHOP-LLC - SingleHop LLC, US	1.55%
32244	LIQUIDWEB - Liquid Web, L.L.C, US	1.47%
14061	DIGITALOCEAN-ASN - DigitalOcean, LLC, US	1.10%
14618	AMAZON-AES - Amazon.com, Inc., US	1.04%
13335	CLOUDFLARENET - Cloudflare, Inc., US	1.03%
36351	SOFTLAYER - SoftLayer Technologies Inc., US	0.93%

Table 3.9 IP address distribution by ASN (Combined)

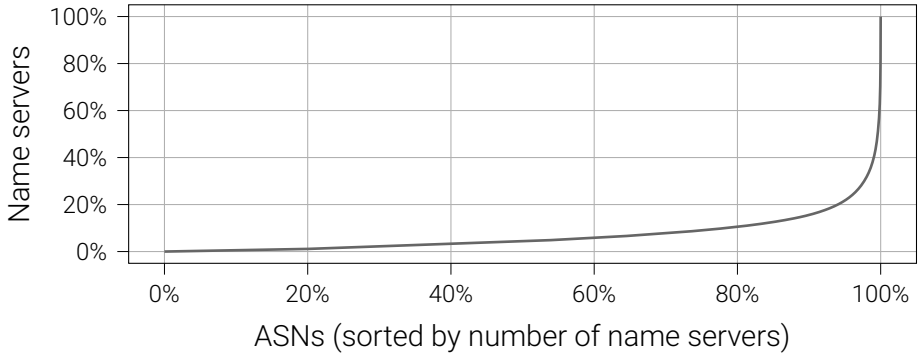


Figure 3.8 CDF for name server distribution

networks. When hosting agents, it is smart to place some inside these countries. By doing so, we can keep the paths short and reduce the attack surface. Remember that there is nothing we can do if the hosting ISP of the name server is malicious. In Table 3.9, we see the top 10 ASN holding most IPs. The distribution of the name servers' IPs is not even. In contrast, 10% of the ASes have 80% of the name servers. In Figure 3.8, we can see that many ASes own only a few name server IPs, and very few ASes own most of the name servers' IPs.

We can see, that in general, the DNS infrastructure is very vulnerable to attacks. At least without any additional security mechanisms. In the following, we will evaluate if we can find independent BGP routes to the name servers even in unideal scenarios. The top one million lists we use are not model examples since they contain many

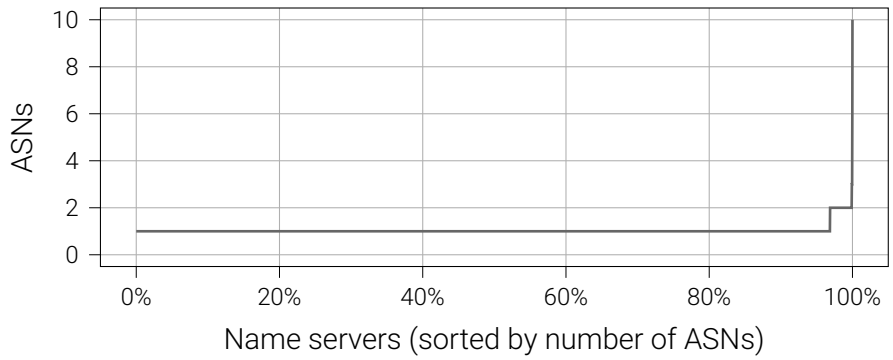


Figure 3.9 Number of ASNs for each name server

misconfigured domains. However, for our demonstration purposes, it is perfect. Showing that something works in an ideal setup is simple. But doing the same for unideal setup is not trivial. In our evaluation, we found that 19% of our list's domains have misconfigurations.

3.6.2.4 Finding multiple paths

Ideally, a name server would use multiple IPs, all located in different ASNs. The reality, however, shows another picture. In Figure 3.9, we can see that almost all name servers use a single AS for all their IPs. We perform an analysis to find out how many unique paths we can find for the name servers of a domain.

We use two sets of ASes. For the first list, we extract the relationship from the CAIDA datasets. After sorting all ASes by the number of customers, we use the 10 ASes with the most customers. For the second list, we used eight independent cloud providers: AWS, Azure, Digital Ocean, GoDaddy, Google Cloud, Inmotion Hosting, Linode, and Vultr. Using these sets, we generate BGP paths, with our previously described simulator, for all name server IPs of a domain. We then count the number of BGP paths we can find that do not overlap. We see in Figure 3.10 that for over 80% of the domain, we have at least two independent paths. These numbers include misconfigured domain names. Some name servers do not reply with an NS record. For these, we can only extract one name server from the SOA record of that domain.

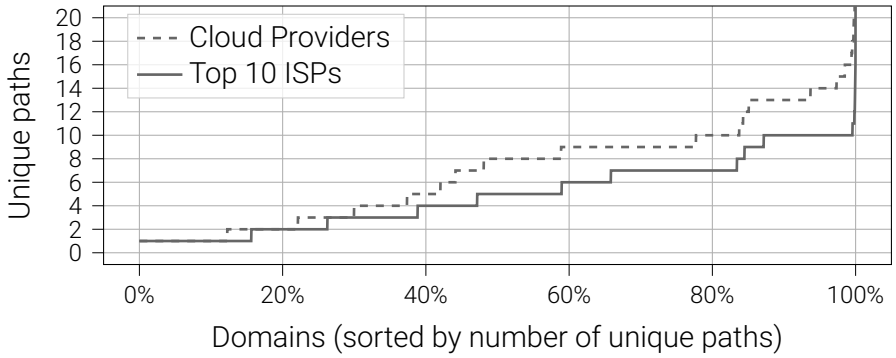


Figure 3.10 Number of unique AS paths per domain (truncated to 20)

3.6.2.5 Anycast IPs

Anycast IPs look like ordinary unicast IP addresses. However, they do not route to the same endpoint. One example of this is the DNS server from Google, which is 8.8.8.8. Someone accessing this IP address in Canada connects to a different server than someone from, e.g., Australia. If both connect to the physically same server, it would be impossible to provide an acceptable latency for both. Anycast is a well-known method for load-balancing and reducing latency. DDV profits from anycast IP addresses. Since the routing is different for agents in multiple regions, it becomes harder to attack the servers. First, an attacker may not know the exact server the agent uses due to load balancing. And second, successfully attacking multiple servers makes the task even harder.

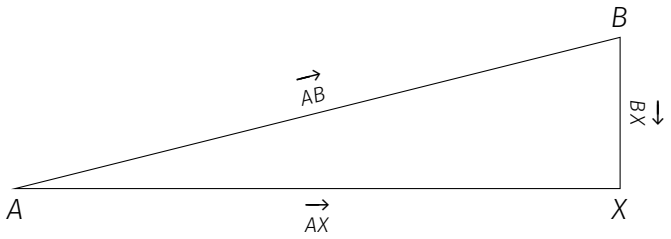


Figure 3.11 Round-trip time triangulation

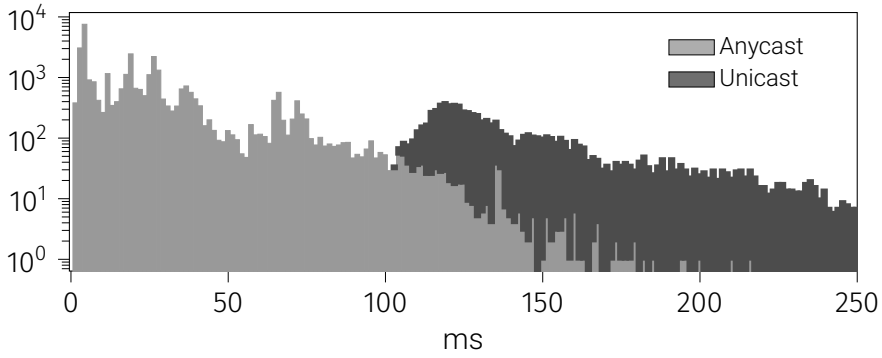


Figure 3.12 Average ping from all vantage points

Since there is no way to tell if an IP is anycast or not, we performed measurements to determine if IPs are anycast. We located servers in the following locations: Amsterdam, Frankfurt, London, New York, Paris, Seattle, Seoul, Singapore, Tokyo, and Toronto. For every name server IP, we measure the round-trip time to each of our servers. We then use triangulation to determine which servers are anycast. We have calculated the distances of all our servers. We use an idealized round-trip time using the speed of light. Let *A* and *B* be two of our servers, and *X* is the target IP. We know that for every target X, the round-trip time for $\overrightarrow{AX} + \overrightarrow{BX}$ cannot be less than $\overrightarrow{AB}$ (see Figure 3.11). If it is, we know that *A* and *B* must connect to two different servers. Since we used ideal round-trip times using the speed of light, the target's connection would be faster than light. We found that out of 11841 IPs, 9945 IPs (84%) are anycast.

In Figure 3.12, we can see the advantage of anycast IPs. The average round-trip time for anycast IPs is much smaller compared to non-anycast IPs. The figure only includes values where all 11 servers succeeded. Otherwise, it would include incorrect averages, e.g., if only servers with a short round-trip time succeeded. However, we can see why so many IPs are anycast; to improve latency.

Ordinary domain validation that CAs already use does not gain any security advantages from anycast IPs. If we know the AS of a CA, we can deploy a server close to them to get responses from the same server. DDV, however, profits from anycast IPs. Even if an attacker knows the AS of every agent, simultaneously attacking all anycast name servers is hard. Our agents rely on non-caching resolvers. Exploiting agents one by one with DNS cache poisoning is not possible.

3.6.2.6 IP prefixes

Strong attackers on the Internet use BGP hijacks. We categorize these attacks as sub-prefix and same-prefix hijacks. Prefixes are a collection of IP addresses designated by the number of leading bits. The traditional notation is the slash notation, e.g., 192.0.0.0/8 would indicate eight fixed leading bits (or the first byte/octet). The remaining 24 bits can change. That means that the prefix covers all IP addresses from 192.0.0.1 to 192.255.255.255, 2^{24} IP addresses in total.

BGP routers on the Internet advertise which prefix they can reach and their distance to it. An ISP can delegate part of its prefix to another AS, e.g., 192.0.2.0/24. Then another AS will advertise this larger prefix (with fewer IPs). When deciding where to route IP packets, there are two rules. First, ASes with the longest matching prefix have the highest priority. Otherwise, ISPs would get the packets for their customers and would have to redirect them themselves. When two ASes have the same prefix lengths, the AS with the shortest distance (fewer hops) has the highest priority. So, when an attacker hijacks a prefix by claiming the same prefix but with fewer hops, we call this same-prefix hijack. However, when an AS advertises a longer, more precise, prefix routers will prefer its route no matter if shorter prefixes are, e.g., directly connected. We call this kind of attack sub-prefix hijacking. However, routers do not advertise or forward advertisements longer than /24. Sub-prefix hijacks are powerful attacks that can steal prefixes globally, while same-prefix hijacks are not harmless but affect local Internet routing.

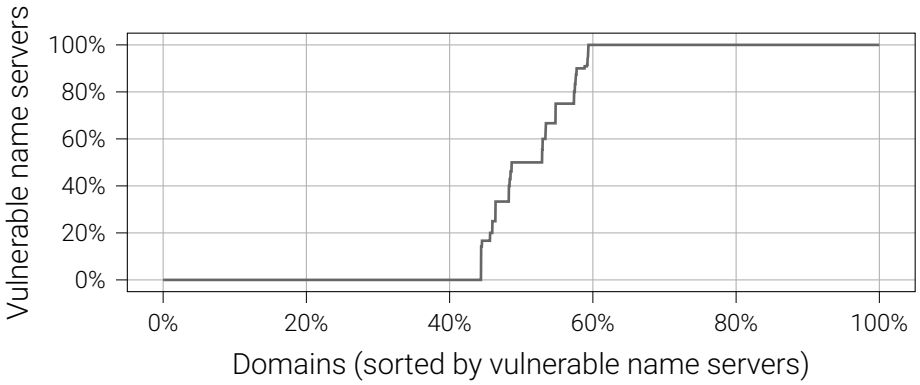


Figure 3.13 Domains vulnerable to sub-prefix hijacks

Sub-prefix hijacks

Name servers that have a prefix shorter than /24 are in general vulnerable to sub-prefix hijacks. We show that many domains use a single prefix and are vulnerable to sub-prefix hijacks. We developed a small tool that resolves the prefixes for the name servers' IPs. We obtained 754857 domains that have at least one name server on a network prefix less than /24. Further filtering for all that have all their name servers on a prefix that is less than /24, there are only 551878 domains on 31,820 prefixes left. We plot the results of our evaluation of the name servers vulnerable to sub-prefix hijacks in Figure 3.13. The X-axis represents the sorted domains. Domains between 60% and 100% correspond to 551,878 that are vulnerable. These have all their name servers on prefixes shorter than /24. The first 45% do not have any vulnerable name servers.

Furthermore, we found that 270,940 domains have all the name servers on one prefix. We plot the distribution of the domains to prefixes in Figure 3.14. We can see that 20% have only one prefix. We found that 201,595 of them have all the name servers on a single vulnerable prefix shorter than /24. We also found that 52,768 domains have a single name server, and 39354 of those are on a prefix shorter than /24. To summarize: 40.67% of the domains are vulnerable to sub-prefix hijacks since all their name servers are on prefix less than /24. 47.10% have at least half of the name servers on prefixes less than /24.

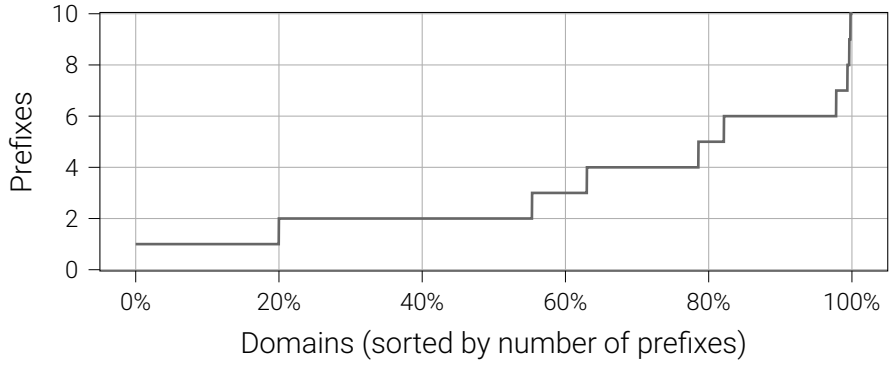


Figure 3.14 Distribution of domains to prefixes

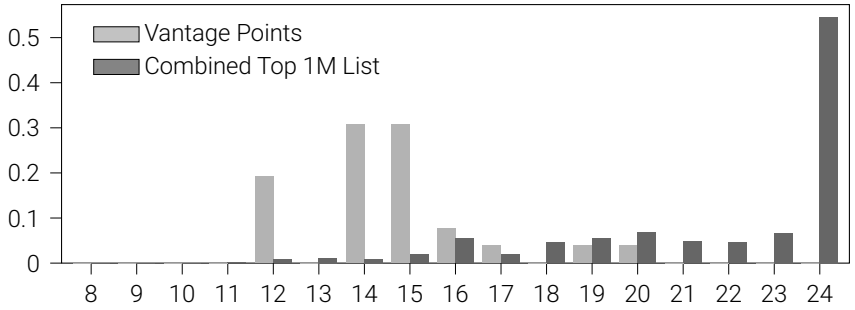


Figure 3.15 IP prefix statistics

We analyzed the agents of Let's Encrypt, or as they call them: vantage points. Let's Encrypt is currently the only CA that uses multiple agents to perform domain validation. In Figure 3.15, we show the prefixes of the vantage points and our combined top 1M list. We can see that all vantage points are vulnerable to sub-prefix hijacking.

The countermeasure to prevent sub-prefix hijacks is simple. By just using /24 prefixes, prefixes cannot be more precise since /24 is the maximum and routers discard /25 prefix announcements. However, a CA can only influence its validation agents and not the target servers. Without enforcing /24 prefixes on target domain name servers, there is not much CAs can do about it. However, sub-prefixes are very noticeable, and routers distribute them. Launching a sub-prefix hijack that remains hidden is impossible.

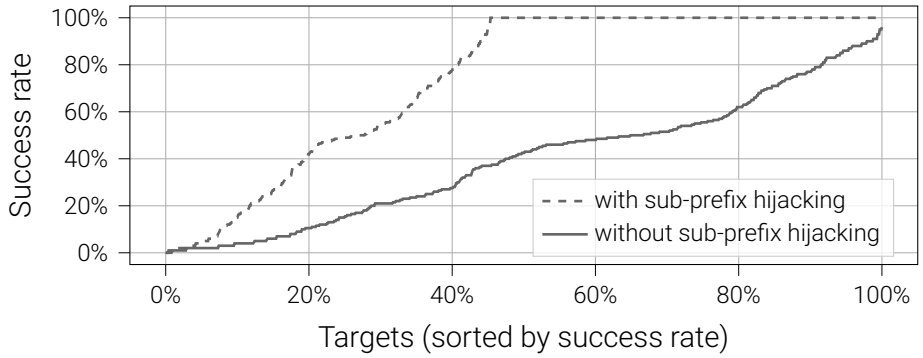


Figure 3.16 Success probability simulation

Same-prefix hijacks

A same-prefix hijack requires proximity. The attacker must be closer to the victim than the agents performing the domain validation. As our example, we used the vantage points of Let's Encrypt again. We put the ASes of the vantage points and the target ASes of the domains from our combined top 1M list into a simulator. Even if Let's Encrypt has multiple vantage points, they all reside in two ASes: AS13649 Flexential (their ISP) and AS16509 Amazon AWS. We assume that Let's Encrypt vantage points use local cache resolvers, and attackers can poison them one by one [59]. We evaluate the probability of a successful attack for same-prefix hijacks and a combination of same-prefix and sub-prefix hijacks using a sample size of 1000 attackers per target. A randomly selected attacker succeeds if it is closer to the victim than the agents. A sub-prefix hijack is successful if the prefix length is shorter than /24.

We can see the results of the simulation in Figure 3.16. The solid line shows attacks that apply same-prefix hijacks, and the dotted line is a combination of same-prefix and sub-prefix hijacks. The simulation demonstrates that launching same-prefix hijacks has a lower success probability than sub-prefix hijacks. In general, networks which are not vulnerable to sub-prefix hijacks are more secure. The plot shows that for 20% of domains, attackers have at least a 60% success probability. Same-prefix hijacks become more difficult as the Internet gets shallower. In Figure 3.17, we see the distribution of hops between the agents and the target AS. There are on average 1.7 hops in-between.

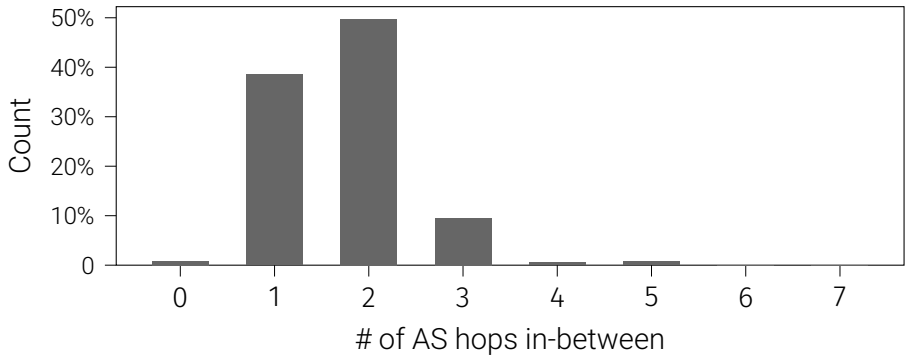


Figure 3.17 Hops in-between Let's Encrypt vantage points and name server ASes

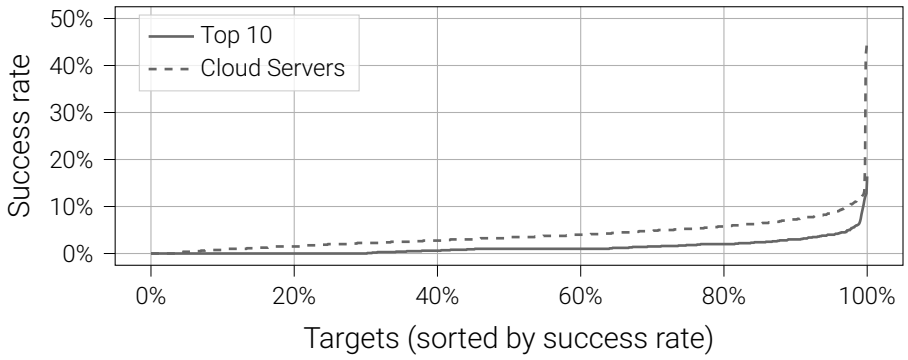


Figure 3.18 Simulating attack success rate

3.6.2.7 Security evaluation of DDV

We previously showed how prefix lengths and BGP paths influence the security of domain validation. We choose 1000 random target ASes and calculate the BGP paths from all agents. We do this for the top 10 ISPs and the cloud providers using our simulator. For each target, we select 100 random attackers ASes and compare the path lengths from the attacker to the target. If the attacker's path length has fewer hops than the agent's path length, we count this as a success for the attacker. So, if an attacker has a shorter path than 50% of the agents, it has a 50% success rate. We then take the average of all 100 attackers ASes. In Figure 3.18 we show the result of our attack simulation. The X-axis is sorted by success rate, i.e., it is not cumulative. We can see that by using carefully selected agent ASes, we improve the security of domain validation.

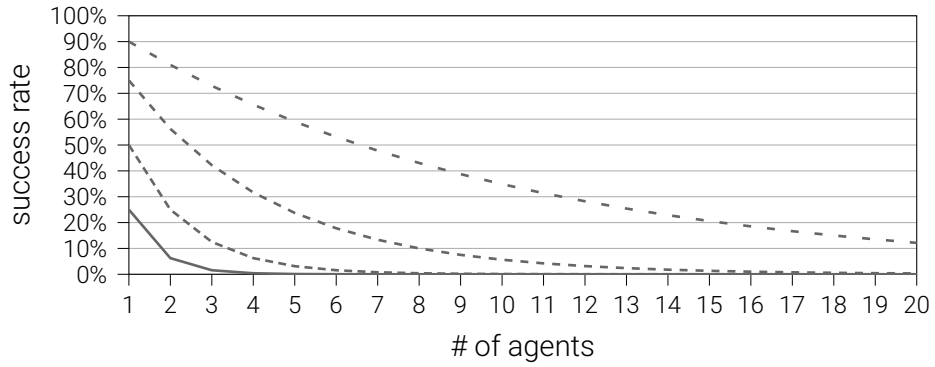


Figure 3.19 Simulation of the success rate of an attacker

Of course, this is under the assumption that every attack succeeds. If we assume that the success probability of an attack is not 100%, we must scale the simulation. If an attacker has a success rate of 90% for one agent, it will have 81% for two agents and 72.9% for three agents. In our setup, all agents' resolvers must not cache results. To attack the domain validation, an attacker must attack all agents simultaneously. Cache poisoning one by one is not possible. In Figure 3.19, we can see that with enough agents, attacks become almost impossible.

3.6.2.8 Selecting agent ASNs

We previously showed simulations using Top 10 ISPs and Cloud providers. However, we still need a method to define ASNs that are good for agents. The logic behind this is simple: connections with fewer hops are less vulnerable to same-prefix hijacks. Ideally, the source and target AS have a direct link. We determine the best ASes for use as an agent by running a simulation using the Caida dataset. We extract the name server ASNs of our combined list of Alexa, Cisco, and Majestic rankings. We determine how far ASes in the Caida dataset are from the name servers. If an AS is a direct neighbor, i.e., directly connected, we award it one point. If an AS has a neighbor with a direct connection to the target AS, we add 0.1 points. We limit the simulation to neighbors of neighbors. We then rank the ASes by their score to make a list of best-connected ASes that as suitable for multiple agent validation as shown in Figure 3.10.

ASN	Name
6939	HURRICANE, US
174	COGENT-174, US
3356	LEVEL3, US
24482	SGGS-AS-AP SG.GS, SG
267613	ELETRONET S.A., BR
51185	ONECOM-AS, GB
7713	TELKOMNET-AS-AP PT Telekomunikasi Indonesia, ID
39120	CONVERGENZE-AS ISP services in Italy, IT
58511	ANYCAST-GLOBAL-BACKBONE Anycast Global Backbone, AU
28186	ITS TELECOMUNICACOES LTDA, BR
41327	FIBERTELECOM-AS Fiber Telecom S.p.A., IT
13786	SEABRAS-1, US
14840	BR.Digital Provider, BR
28634	Life Tecnologia Ltda., BR
25091	IP-MAX, CH
38880	M21-AS-AP Micron21 Datacentre Pty Ltd, AU
37468	ANGOLA-CABLES, AO
56665	TANGO-TELINDUS, LU
49605	DTS-AS DTS, IT
13237	LAMBDANET-AS European Backbone of AS13237, DE
1299	TELIANET Telia Carrier, EU

Table 3.10 Top 20 ASNs for agent selection

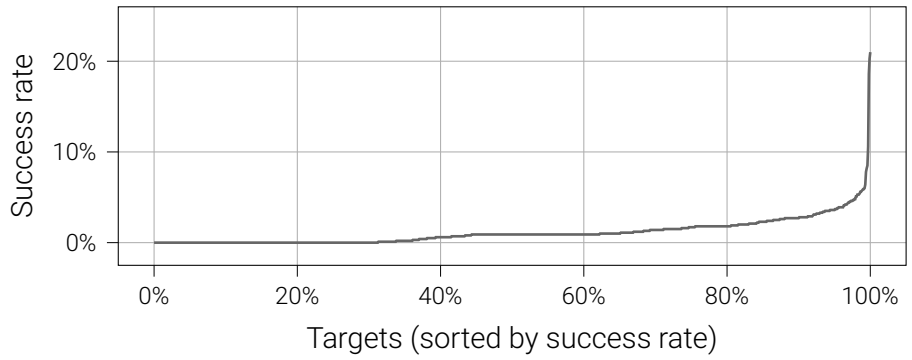


Figure 3.20 Simulating attack success rate of top 10

We performed the simulation we already did in Figure 3.18 for the top 10 of our calculated best ASNs. We can see in Figure 3.20 that they perform like the top 10 ISPs we previously evaluated. They are so similar that it is hard to plot them both. With extreme magnification, we can see that our calculated performs 0.1-0.2% better.

However, the next question that arises might be: How long is this calculation valid? We evaluated the consistency of the AS distribution of DNS name servers. We used our combined list and extracted the name servers. For each domain name, we resolve its name servers and the IP address of these name servers to look up the corresponding AS. We did this in April 2020 and December 2020. We see that the AS changes for 13.41%. Running our algorithm to determine the best agent ASes, the top 34 ASes remained unchanged. In the top 50, only place 35 and 36 switch places. We assume that the calculations last for a few years. The name servers switch from DNS and VM providers to other VM providers, e.g., Cloudflare DNS with DDoS protection. Table 3.11 shows where most servers left while Table 3.12 shows where they switched to.

ASN	Name
394695	PUBLIC-DOMAIN-REGISTRY - PDR, US
32475	SINGLEHOP-LLC - SingleHop LLC, US
36351	SOFTLAYER - SoftLayer Technologies Inc., US
24940	HETZNER-AS, DE
16276	OVH, FR
19527	GOOGLE-2 - Google LLC, US
55002	DEFENSE-NET, US
26496	AS-26496-GO-DADDY-COM-LLC - GoDaddy.com, LLC, US
46606	UNIFIEDLAYER-AS-1 - Unified Layer, US
51559	NETINTERNET Netinternet Bilisim Teknolojileri AS, TR
29169	GANDI-AS Domain name registrar, FR

Table 3.11 ASNs from which name server went away (top 10)

ASN	Name
13335	CLOUDFLARENET, US
15169	GOOGLE - Google LLC, US
24940	HETZNER-AS, DE
16509	AMAZON-02, US
16276	OVH, FR
19871	NETWORK-SOLUTIONS-HOSTING, US
14061	DIGITALOCEAN-ASN, US
40034	CONFLUENCE-NETWORK-INC, VG
14618	AMAZON-AES, US
209453	GANDI-LIVEDNS AS for LiveDNS Anycast servers, FR
32244	LIQUIDWEB, US

Table 3.12 ASNs to which name servers changed (top 10)

3.7 Conclusion

PKI plays a critical role in Internet security. Automated domain validation is still the most efficient and easiest-to-use method for ownership verification. Unfortunately, the comfort comes with the downside of being less secure, making it vulnerable against off-path attacks. We demonstrate how we successfully obtained a certificate using these attacks.

To evaluate same-prefix hijacks, we developed a BGP simulator for large-scale evaluations. The simulator is different from the simulator we initially used in [36]. It is more complex and was the first to incorporate relationship-based search and performance optimizations. Other simulators, like our initial simulator, use breadth-first search. We demonstrate that our simulator achieves higher accuracy and better performance in contrast to existing simulation platforms. Using the simulator, we evaluate the effectiveness of DDV. We show that securing the CAs does not suffice for ensuring the security of domain validation in PKI. Our simulations and measurements show that the domains are not resilient to attacks against domain validation and introduce a weak link in the PKI ecosystem.

Deployment of cryptography is far from being usable. Instead of relying on using it, we need another way to bootstrap security on a PKI. We utilize the distributed nature of the Internet and make assumptions about an attacker's abilities. We propose a distributed domain validation as a drop-in replacement for current domain validations. We evaluate the security through our simulations and can prove that it is resilient against strong MitM attackers. We show that many IP addresses are anycast. In contrast to the standard domain validation, the distributed domain validation benefits from anycast IPs.

We also evaluate how to distribute validation agents on the Internet. We provide simulations and a method the rate how good ASes are for agent placements. We also provide analysis on the longevity of our calculations.

CHAPTER 4

Randomness

Without random keys and nonces, the best cryptographic algorithms become useless. Humans are miserable at generating random sequences. However, we take it for granted that randomness is always available. Cryptographic algorithms usually mention taking random values and nonces. However, cryptographically secure randomness is not trivial. It does not matter how robust the cryptographic algorithms are. If an attacker can predict random bits, all security falls. We want to look at a building block of cryptography.

4.1 Introduction

Randomness is crucial to cryptography. Randomness plays a critical role in the design of security mechanisms. Many applications rely on unpredictable sequences of random bits: cryptographic keys, SSL/TLS nonces, initialization vectors for symmetric encryption, password generation, sequence numbers and ports in operating systems, address space randomization for preventing memory corruption vulnerabilities, and many more. Typically, cryptographers assume that perfect randomness is available. However, access to perfect random bits is non-trivial in practice. We usually generate randomness with a Pseudorandom Generator (PRG). We collect random seeds and expand them into a longer pseudorandom string.

4.1.1 Randomness generation

Generating randomness is a longstanding problem. Despite the critical role that secure generation of unpredictable pseudorandom bits plays, there is a long history of attacks exploiting bugs and vulnerabilities in PRGs, e.g., [32, 50, 68, 79, 84, 85, 146]. The causes for vulnerabilities in the generation of pseudorandom strings are often due to faulty implementations or the reuse of entropy. Incorrect usage, as well as wrong assumptions, can create vulnerabilities. E.g., one uses a system developed for a setup with access to entropy sources which are not present (like keyboard or mouse inputs on routers or cloud platforms). Randomness failures and vulnerabilities are a severe problem in practice. There is a large body of work to improve the randomness generation, but bugs and benign failures can always persist, and malicious adversarial strategies are hard to foresee and counter. Furthermore, reliance on a single pseudorandom generator is risky due to potential intentionally inserted backdoors [61].

4.1.2 Contribution

We propose an alternative approach for generating pseudorandom strings. We leverage the distributed nature of the Internet for collecting randomness from public services on the Internet. Based on this methodology, we develop our Distributed Pseudorandom Generator (DPRG) and demonstrate how it guarantees security against strong practical attackers and how it addresses the main shortcomings in existing PRGs. Specifically, it allows an automated generation of pseudorandom

strings without the dependency on entropy sources or user input. It provides robustness because it does not rely on a single system that may be faulty or vulnerable. Failures generated by an attacker do not subvert the security of the randomness generation if at least one of the servers securely provides a good pseudorandom string. We establish connections to well-managed servers on the Internet and collect randomness from them. The bitstrings received from some servers may be insecure, e.g., not random or known to the attacker. Indeed, such incidents are not far-fetched. Adrian et al. [5] showed how to compute discrete logs of 7% of Alexa servers that use weak Diffie-Hellman parameters or maintain support for obsolete export-grade cryptography. Our DPRG uses AES encryption in CBC mode and an HDKF to extract randomness and inputs for handshakes. To predict our randomness, an attacker would have to reverse the hash function and know all previous values of the AES CBC cipher since all ciphertexts depend on these. Hence, the output is secure (i.e., pseudorandom) if, at least, one received bitstring is secure.

We analyze the distribution of different randomness sources like HTTP, SMTPS, SSH, and TOR and present an implementation of DPRG using the TOR network. We analyze the quality of randomness and performance of our DPRG and show that we can achieve highly secure randomness only from user space.

4.2 Background

In this section, we want to clarify terms and introduce the cryptographic primitives we need. We also introduce TOR, which we use for our reference implementation since we assume, that not all readers know about the inner workings of TOR and its Public Key Infrastructure (PKI).

4.2.1 Randomness

The definition for the term randomness depends on the application field. In graphics or audio, randomness describes a noise pattern. It does not matter if one can predict these sequences since their goal is to look and sound random. In statistics, when performing evaluations with random data, we even want the outcome to be the same every time we run it. In cryptography, however, we need unpredictable randomness. We define randomness as not predictable by humans or computers by any means. We cannot achieve true randomness using algorithms and rely on physical effects, e.g., atmospheric noise or radioactive elements. We call the output of a random number algorithm pseudorandom. If we are talking about a Random Number Generator (RNG), we typically talk about a Pseudorandom Number Generator (PRNG). When we have hardware that collects true randomness, we explicitly mark this as a True Random Number Generator (TRNG). There are still generators more suitable for cryptographic uses. A common term for this is Cryptographically Secure Pseudorandom Number Generator (CSPRNG). However, in the cryptographic context, we always assume that a PRNG is safe for cryptography. The same applies to this work.

4.2.2 Cryptographic primitives

We assume that the reader knows cryptographic primitives like hash functions, MACs and HMACs, symmetric and asymmetric encryption, and key exchanges. However, we briefly introduce the methods we use: the HMAC-based Extract-and-Expand Key Derivation Function (HKDF), AES in CBC mode, and X25519.

A HKDF extracts or expands a pseudorandom key using a Hash-based Message Authentication Code (HMAC) function [114]. The output is irreversible due to the hash function. It improves random bit sequences with biases.

The Advanced Encryption Standard (AES) is ubiquitous. Many CPUs provide hardware acceleration for it. We use it in the Cipher Block Chaining (CBC) mode, where the output of a previous block is XORed with the input of a new one. This mode prevents that encrypting equal plaintexts does not yield the same ciphertext.

X25519 [26] is a Diffie-Hellman key exchange based on the elliptic curve, called Curve25519 [27], by Daniel J. Bernstein. Curve25519 is by far the most popular elliptic curve. Popular applications like Signal, Matrix, and Wireguard rely on it.

4.2.3 TOR

The onion router project [65], from which the name TOR derives, first released their proxy in September 2002 [64]. It is a mix-network that enables anonymous end-to-end communication. It routes the traffic over multiple routers and encrypts it with multiplication encryption layers. A client negotiates encryption keys with each router on the circuit (that is what TOR calls the paths), and only one router can decrypt one of the layers. Each router forwards the traffic without revealing the IP address of the incoming packet. So, e.g., the second router will not know the client's IP address, and the third router will not know the first router's IP address. The TOR authorities perform the coordination of all routers. There are ten TOR authorities: bastet, dannenberg, dizum, faravahar, gabelmoo, longclaw, maatuska, moria1, serge, and tor26. However, we will only consider nine since Serge is a bridge authority. It is a connection for bridge relays that enables users with limited Internet, e.g., IPs blocked by the government, to access the TOR network. TOR hides bridge relays. The authorities, excluding serge, compile a list of all routers called consensus. Currently, the TOR network has about 6500 routers.

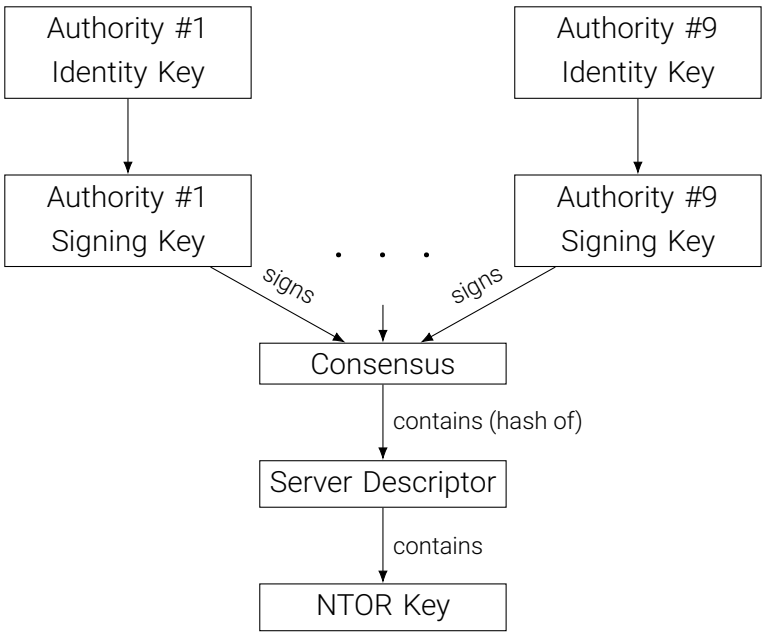


Figure 4.1 Chain of trust

4.2.3.1 Consensus

The nine authorities agree on a consensus. This consensus contains the list of all routers available to the TOR network. It is publicly available [153] as a download using standard HTTP. Every authority signs the consensus with its signing key, as shown in Figure 4.1, so even when downloading over unencrypted HTTP, the client can verify the integrity. For each router in the consensus, there are several details like the nickname, IP address, port, the hash value of the identity RSA public key, and the hash of the most recent descriptor. The descriptors provide further information about a router. Of course, the collection will only be as trustworthy as the PKI used.

4.2.3.2 Keys and descriptors

The hash value of a node's RSA public key forms its identifier. TOR only uses intermediate keys, and the sole purpose of the public key is to sign these. The identity keys of TOR authorities are hardcoded into every TOR client and do not change. Changes within these need changes in every client's software. The TOR authorities use their identity keys to sign the keys they use for their signature for the consensus. The signing keys frequently change. The authority identity keys can form a root of trust.

Authorities gather votes and agree on a common consensus. If we want to verify it, we must download each signing key. We can use the identity key to check the integrity and correctness of a signing key. Finally, we can use the signing keys and check the signatures of the consensus. An attack would need most majority of authorities to succeed. The TOR authorities run geographically spread across the Internet, which makes attacks even harder.

To get more information about a router, a client can download a descriptor for a router using the hash of the descriptor (which is part of the consensus). The descriptor contains more keys, like the identity RSA public key or NTOR key. We need these keys for the circuit creation described later. We can verify a descriptor by comparing the hashed public key with the identity hash of the router. If these match, we use the public key to verify the signature of the descriptor itself.

4.2.3.3 Circuits

To conceal the identity of users, TOR establishes connections over multiple routers. We call this path a circuit, and each circuit consists of three nodes. This value provides a good balance between privacy and network load [154]. The first node of each circuit must be a guard node. Only these nodes will see the IP address of the client establishing a circuit. These guard nodes have requirements for uptime and bandwidth for the TOR network to consider them as guard nodes. To set up these circuits, TOR uses its protocol. The TOR spec [155] describes this protocol. TOR tunnels all messages through a TLS connection. Optionally, one can verify the TLS certificates. However, the tor spec takes care of authentication in their protocol.

TOR embeds every payload in cells. All cells are 514 bytes since version 4. Before this version, it was 512 bytes. There are a few exceptions, like the VERSIONS, CERTS and AUTH_CHALLENGE, which do not need a padding to 514 bytes or can even exceed that size. In the following, we will describe the most important cells for our use case.

VERSIONS

The first cell sent to a router is the VERSIONS cell. The initiator sends this cell containing all TOR versions supported by the initiator. The responder will respond with the highest TOR version it supports or close the connection.

AUTH_CHALLENGE, AUTHENTICATE, and CERTS

After sending a VERSIONS cell, the responder immediately sends the CERTS cell. This cell includes certificates the responder uses. Unfortunately, this does not include the NTOR key. The AUTH_CHALLENGE cell includes data an initiator needs to authenticate itself. E.g., it includes 32 random bytes, which the initiator can sign to prove its identity and send via an AUTHENTICATE cell. The TOR spec defines that "Responders MUST generate every challenge independently using a strong RNG or PRNG." [155]. The CERTS cell contains all keys a router supports.

NETINFO

The NETINFO cell only contains a timestamp, the "own" IP addresses, and the "others" IP address. IP addresses can be both IPv4 and IPv6. However, only a few routers support IPv6. First, the responder sends this cell with the initiator. The responder also sends a NETINFO cell to conclude the initial setup.

CREATE, CREATE2, and NTOR handshake

To finally create the circuit, there are two cells: the older CREATE cell and the newer CREATE2 cell. We only care about the newer CREATE2 which replaces the old one. We can choose from two handshakes: the original TAP handshake and the newer NTOR handshake. The NTOR handshake is faster and better suited for low-power devices because it uses elliptic curves with shorter keys. The CREATE2 cell will contain the first half of the handshake and the CREATED2 the second half.

In the first half of the handshake, the initiator generates a temporary Curve25519 key pair (x, X). It will then send: the identity key (ID), the NTOR key of the responder (B), and the temporary public key (X) to the responder.

$$secret\_input = X \circ y \mid X \circ b \mid ID \mid B \mid X \mid Y \mid PROTOID,$$

where $X \circ y$ is the dot multiplication of X and y, b is the private key of the responders NTOR key B, $PROTOID$ is a constant variable containing the string "NTOR-curve25519-sha256-1", and $\mid$ is the concatenation operator.

Further, the responder will calculate

$$\begin{aligned} KEY\_SEED &= \text{HKDF}(secret\_input, t\_key) \\ verify &= \text{HKDF}(secret\_input, t\_verify) \\ auth\_input &= verify \mid ID \mid B \mid Y \mid X \mid PROTOID \mid \text{"Server"} \\ AUTH &= \text{HKDF}(auth\_input, t\_mac), \end{aligned}$$

where HKDF is a hash-based message authentication code using the constants t_key, t_verify and t_mac as a key to hash the $secret\_input$.

In the CREATED2 cell, the responder will include Y and $AUTH$. When the initiator receives the cell, it can calculate

$$secret\_input = Y \circ x \mid B \circ x \mid ID \mid B \mid X \mid Y \mid PROTOID$$

and calculate KEY_SEED and $AUTH$. $AUTH$ and uses it to verify the identity of the responder since only a router in possession of the private key b can calculate the correct $secret\_input$.

RELAY, EXTEND, EXTEND2, EXTENDED, and EXTENDED2

After establishing the connection to the first node, both sides derive an encryption key from the KEY_SEED. The initiator will use EXTEND or EXTEND2 cells to extend the circuit, i.e., add another router to the end of it. The EXTEND and EXTEND2 cells are like the CREATE and CREATE2 cells and hold the first half of the handshake. The EXTENDED and EXTENDED2 contain the second half of the handshake. The responder of the connection will forward these cells to another router.

DESTROY

To close connections, TOR uses a `DESTROY` cell. The only content is the reason for the teardown, which includes options like protocol violations or timeouts.

4.3 Related Work

Attackers use high-profile PRG failures and vulnerabilities for exploits. The attacks against PRGs can lead to exposure of private keys of cryptographic systems, e.g., low entropy can allow recovery of plaintext and enable attackers to predict the ephemeral Diffie Hellman session keys [79,137].

One significant vulnerability is insufficient entropy. During boot or when randomness pools in operating systems are exhausted, there might not be enough entropy. Another example is when Linux reaches a global file descriptor limit, and no process can access the system randomness via /dev/random. Most applications then proceed without the randomness from the operating system. In that case, cryptographic libraries produce vulnerable keys, which can potentially affect the security of multiple applications. In particular, [57] showed that components of OpenSSL, including SSL/TLS pre-master secret generation and RSA key generation, as well as the arc4random function used for cryptographic randomness in FreeBSD, OpenBSD, and macOS, are all affected by this issue. [66] also found that /dev/random and /dev/urandom do not accumulate enough entropy and presented attacks against them. These vulnerabilities allow an attacker to predict a future value based on the previous outputs. An attacker might even retrieve the seed value. Debian, e.g., had a severe bug in its random number generator. A code linter complained about an uninitialized value. When developers removed the affected line, the random number generator only used the current process id as its seed [44]. The seed was effectively limited to 32,768 values (which is the maximum process id).

Previous research showed that system PRGs provide poor security in virtualized environments [75,76,108]. [137] performed reset vulnerabilities and demonstrated that a user-space process such as TLS could suffer significant loss of security when run in a VM that resumes multiple times from a snapshot. [75] showed that resets could lead to exposure of secret keys generated after snapshot resumption. Among other factors causing the weaknesses, the software entropy sources are weaker in the virtualized environment, e.g., due to lack of mouse and keyboard inputs. However, vulnerabilites affect even non-virtualized environments. In 2019, AMD's Ryzen processors microcode had a flawed RDRAND() function [3]. The function, which is supposed to return random values from its hardware random number generator, reported the same value on every call.

A large-scale Internet measurement by [93] of SSH and TLS keys generated by headless or embedded systems and server management cards found vulnerabilities in cryptographic keys caused by insufficient entropy in inputs to PRGs. The causes for the problems were faulty implementations that generate keys automatically on the first boot without having collected sufficient entropy. [146] showed vulnerabilities in DNSSEC keys generation in well-established large registrars and DNS hosting providers and traced the problems to reuse of cryptographic material in attempts to save on randomness.

A related research direction attempts to weaken dependence on PRGs in practice by amplifying randomness [35,56]. There are also centralized services that use other sources of entropy to generate random bits, such as HotBits [106], which uses the uncertainty in quantum mechanical laws of nature, or LavaRnd [118], which uses variation in the timing of hardware interrupts. The is also the infamous example of Cloudflare using the photos of 100 lava lamps to generate randomness [54].

Previously, researchers identified the benefit of distributed servers to ensure resilience even in the presence of some corrupted servers, e.g., [51,150]. In contrast to our mechanism, these works propose setting up dedicated servers with an initial cryptographic setup (including a key shared between the servers) for the collective generation of randomness. The main limitation of this approach is the requirement to set up servers and ensure independent third parties. In contrast, our mechanism does not require any of these. We also do not assume a shared key or string between the servers. Furthermore, [51, 150] requires that a majority of the servers are not corrupted for the overall security to hold. For our approach, it suffices that one server is uncorrupted.

4.4 Distributed Pseudorandom Generator

A DPRG utilizes the randomness that already exists on the Internet without relying on a single source. It connects to multiple Internet services and extracts shared secrets from cryptographic handshakes.

Our initial concept collected three random strings from individual servers. We would XOR all three and use the output as our randomness. Under the assumption that at least one connection is not malicious, we consider our randomness to be random. We also had the requirement that we must trust the local ISP. We analyzed the BGP paths for each connection and avoided overlaps. This design has two drawbacks. First, we waste random strings, two-thirds, to be precise. Second, we must trust our ISP. So, we improved the initial concept.

To extract a random string, we perform a handshake that contains a key exchange, like Diffie-Hellman. However, we then feed the result into an AES CBC encryption. To predict the current state of the cipher, an attacker must know all inputs (since in CBC mode, the inputs are XORed with the previous outputs). Now we can use all random strings we collect. However, since we need some randomness for the key exchange itself, we extract some randomness by using the HKDF on our current ciphertext. Reversing the HKDF is not possible, so the current state of our cipher is always secure. Now we can continuously feed our AES encryption with randomness. We can also expand our random strings by calling the HKDF multiple times. Everything we extract from our DPRG must go through a HKDF to keep the current cipher state always hidden, even to programs running locally and accessing the randomness. DPRG keeps a buffer and collects new randomness when the pool is empty.

4.4.1 Sources

DPRG can use all kinds of services that provide a key exchange. For our evaluation, we chose public services that perform a cryptographic handshake. Our first source is the Alexa Top 1M list. Alexa Internet (a subsidiary of Amazon) publishes a ranking of the top one million websites [6]. Their goal is to estimate the popularity of websites by using traffic data acquired by Alexa and links to these sites. This list is by far the most popular source for website lists in research, which is why we include it. However, even if it names implies one million web sites, the ranking usually has less than one million websites, and many of the websites do not work anymore. We also

	Alexa	HTTPS	SMTP(S)	SSH	TOR
AFRINIC	0.67%	3.37%	3.54%	5.27%	0.38%
APNIC	9.68%	13.85%	21.80%	21.83%	3.51%
ARIN	55.81%	57.04%	34.18%	44.01%	26.81%
LACNIC	1.47%	3.05%	4.05%	3.96%	1.42%
RIPE NCC	32.48%	22.69%	36.43%	24.94%	67.88%

Table 4.1 RIR distribution

included routers from the TOR consensus. TOR has a custom handshake protocol that we can use to extract random strings. Lastly, we use data from scans.io [149]. We select services with key exchanges, i.e., HTTPS, SMTP (with STARTTLS), STMPS, and SSH. These are just IP scans without any ranking.

To extract information like RIRs and countries, we must map the IP addresses to ASNs. For the mapping, we use the online service provided by Team Cymru [152]. We map each address to its ASN and then evaluate the distribution of the IP addresses using this data. Our set contains 429.463 (Alexa) and 6531 (TOR) unique IP addresses. For the IPs from scans.io, we reduced the IPs to a 2.5 million per service. Otherwise, resolving, e.g., 50 million HTTPS IP addresses to ASNs is too time-consuming.

There are five Regional Internet Registries (RIRs) that operate in different regions: African Network Information Centre (AfriNIC), Asia-Pacific Network Information Centre (APNIC), American Registry for Internet Numbers (ARIN), Latin America and Caribbean Network Information Centre (LACNIC), and Réseaux IP Européens Network Coordination Centre (RIPE NCC). Table 4.1 shows the RIR distribution of our data sets. We can see that ARIN and RIPE NCC own most of the IP addresses. Usually, ARIN is ahead of RIPE NCC except for TOR, where RIPE NCC owns 67.88% of the IP addresses.

Alexa	HTTPS	SMTP(S)	SSH	TOR
US (54.36%)	US (55.11%)	US (31.96%)	US (42.35%)	US (22.17%)
RU (6.41%)	DE (4.51%)	DE (7.19%)	CN (12.32%)	DE (21.10%)
DE (5.97%)	CN (3.85%)	AU (6.39%)	DE (5.05%)	FR (12.92%)
FR (2.71%)	SC (2.48%)	FR (5.87%)	FR (4.03%)	NL (5.16%)
TR (2.69%)	GB (2.47%)	JP (4.19%)	SC (3.77%)	CA (4.81%)
GB (2.06%)	FR (2.38%)	CN (4.12%)	RU (2.56%)	GB (3.48%)
NL (1.99%)	CA (2.17%)	PL (3.89%)	AU (2.10%)	RU (3.15%)
CA (1.73%)	JP (2.03%)	GB (3.34%)	CA (2.07%)	SE (2.19%)
IR (1.41%)	AU (1.48%)	HK (2.94%)	GB (2.04%)	CH (2.17%)
UA (1.40%)	NL (1.48%)	CA (2.51%)	NL (1.88%)	RO (1.55%)

Table 4.2 Top 10 country distribution

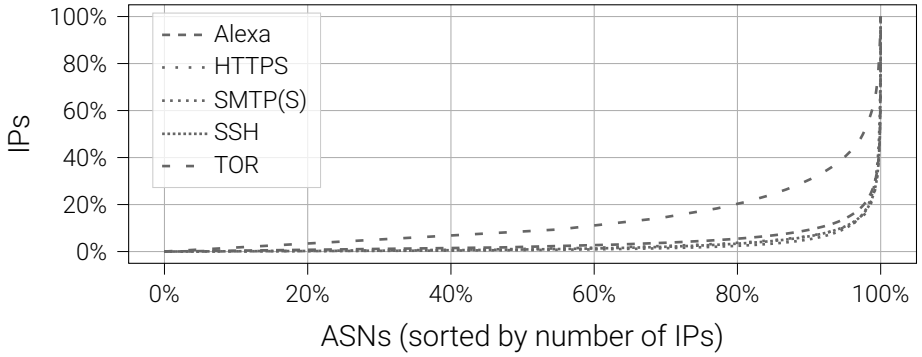


Figure 4.2 IP to ASN distribution

Table 4.2 shows the geographic distribution by country. The US owns most of the IP addresses for all our data sets, which is not too surprising, considering that the US has the most computers in the world [116]. Figure 4.2 shows the distribution of IP addresses to ASNs. The Y-axis shows the cumulative number of IP addresses. The X-axis shows the ASNs sorted by the numbers of IPs they own. We can see that for the Alexa, HTTPS, SMTP(S), and SSH IPs, about 3-4% of the ASes own 80% of the IPs. We see that Alexa's distribution is slightly better and that HTTPS, SMTP(S), and SSH have almost the same distribution. However, TOR has a much better distribution, and only 20% of the ASes own 80% of the IP addresses.

ASN	Name	IPs
13335	CLOUDFLARENET, US	16.79
16509	AMAZON-02, US	7.22
14618	AMAZON-AES, US	3.79
24940	HETZNER-AS, DE	3.26
14061	DIGITALOCEAN-ASN, US	3.08
16276	OVH, FR	2.70
46606	UNIFIEDLAYER-AS-1, US	2.66
15169	GOOGLE, US	1.93
26496	AS-26496-GO-DADDY-COM-LLC, US	1.86
63949	LINODE-AP Linode, LLC, US	1.22

Table 4.3 Top 10 ASN distribution for Alexa

ASN	Name	IPs
16276	OVH, FR	9.55
24940	HETZNER-AS, DE	6.28
53667	PONYNET, US	4.74
12876	Online SAS, FR	4.05
14061	DIGITALOCEAN-ASN, US	2.78
63949	LINODE-AP Linode, LLC, US	2.28
3320	DTAG Internet service provider operations, DE	2.06
197540	NETCUP-AS netcup GmbH, DE	1.83
6830	LGI-UPC, AT	1.60
51167	CONTABO, DE	1.41

Table 4.4 Top 10 ASN distribution for TOR

Table 4.3 and Table 4.4 show the top 10 ASNs for Alexa and TOR. We see that many web pages in the Alexa list use Cloudflare or Amazon to host their sites. The list for TOR includes many VM or colocation hosters like OVH, Hetzner, Digital Ocean, Linode, and PonyNet. For the HTTPS list from scans.io, Akamai takes the top place. We added the tables for scans.io to the appendix.

Overall, we can say that the TOR network is a suitable choice to collect random strings. The distribution of the IPs is better than the other lists we compared. For potential attackers, it is harder to manipulate our random string collection. However, with enough rounds, other lists become just as secure as the TOR list. We must only connect to one uncompromised router.

4.5 TORC

To demonstrate our DPRG, we created a reference implementation using the TOR network, which we call TOr Randomness Collector (TORC). We previously showed that the TOR network is a suitable network for randomness collection. TOR router IPs are not as concentrated on single ASes as other IP lists we used. Also, the TOR network does not have a single root of trust. The TOR authorities determine all decisions by voting. Unlike the PKI used for web pages where a single Certificate Authority can decide if a certificate is valid or not, in TOR, there is a consensus of all routers. Also, many people trust the TOR network because of its decentralized structure and excellent privacy. Overall, we think TOR is a perfect candidate for our reference implementation.

TORC is a small and lightweight open-source tool to collect random data using the TOR network. TORC can either output a given amount of data or write to a pipe, like `/dev/random`. TORC fills a buffer and returns random values from that buffer to accelerate randomness collection. It automatically refills that buffer in the background. We can set every configuration value in the `torc.conf` configuration file (see Listing 4.1). To compile TORC, we only require OpenSSL V1.1 or higher as a dependency. TORC is cross-platform and runs on Linux and macOS.

```
ip=127.0.0.1 // IP and port to download
port=80      // consensus and descriptors
routers=5    // # of router to use for initialization
expand=1     // Randomness expansion factor
buffer=65536 // Pipe buffer size
```

Listing 4.1 Example torc.conf configuration file

4.5.1 Components

TORC consists of three main components: an AES cipher in cipher block chaining (CBC) mode, an HMAC-based Extract-and-Expand Key Derivation Function (HKDF) [114], and a Diffie-Hellman key exchange based on the Curve25519 [27], also called X25519 [26]. The AES CBC cipher will ensure randomness by encrypting collected randomness with the previous cipher block. We use the X25519 key exchange to extract the KEY_SEED and the HKDF to extend the randomness.

```
function InitCipher(seed)
  x_1, x_2, x_3, x_4 ← divide seed into 32 bytes chunks
  iv ← HKDF(x_1, ":AESInitVector")
  key ← HKDF(x_2, ":AESInitKey")
  router_seed ← HKDF(x_3, ":RouterSeedInit")
  init AES with key, iv
  AES(x_4)
end function
```

Algorithm 4.1 Initialize cipher

4.5.2 Initialization

Before we can collect any randomness, we need an initial seed. If we fully trust our local ISP, we can use a timestamp. In that case, we would assume that at least one connection is good. We require this seed only once and it will automatically update after every initialization with a new value. We fetch all authorities signing keys. Then we download and verify the consensus using these keys and select random routers. The router selection does not need cryptographically secure randomness. We want to prevent all TORC clients from connecting to the same routers. Instead, we want to distribute all initial requests. We hash a timestamp using the HKDF and generate three 256-bit values. We use these values as the AES initialization vector and key, and an initial $router\_seed$. Then, we collect a 32 bytes (256 bits) KEY_SEED from at least five routers. One can change this value in the configuration file of DPRG. TORC will encrypt every KEY_SEED without using the ciphertext output. With every encryption, the AES CBC cipher will permutate. If at least one handshake was not malicious, it is impossible to know the last cipher block. We can see an example of the initialization in Algorithm 4.1.

```
function GetRandomRouter()
    max ← n_routers · (65535\n_routers)
    while true do
        R_seed ← AES(R_seed)
        R_hash ← HKDF(R_seed, ":RouterSelect")
        R ← router_hash[30:31]
        if R > max then
            continue
        end if
        return R mod n_routers
    end while
end function
```

Algorithm 4.2 Random router selection

4.5.3 Router selection

To select a router, we encrypt a 256-bit $router\_seed$ value. Then, we hash the value using the HKDF and use the last two bytes. The TOR network has about 6500 routers which allow us to represent them using two bytes. We do not need any padding for the AES 256-bit encryption because the $router\_seed$ is also 256 bits. For an equal distribution, we split up the 65536 (two bytes) values into the number of TOR routers and only use numbers inside a block big enough for all routers. If e.g., there are 6500 routers, we would split up the 65536 into 65000 and skip all numbers that are larger than 65000. We can repeat this step over and over to get a random router. Algorithm 4.2 further illustrates the router selection we described.

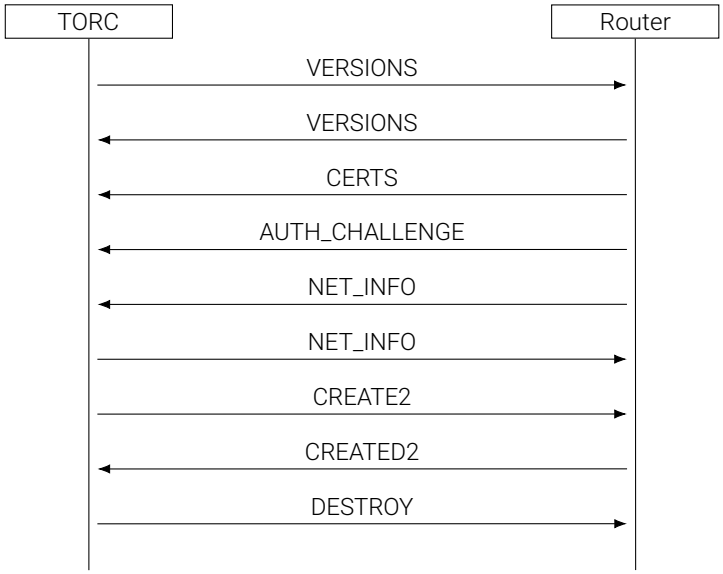


Figure 4.3 TORC circuit creation

4.5.4 Collecting randomness

To collect randomness, we create a TOR circuit for a randomly selected router. We use the standard C rand() function to set the circuit ID, which does not need secure randomness. We could use the same circuit ID for every circuit for our purpose. However, a static circuit ID would make our collector easily identifiable. We extract the 32 random bytes from the *AUTH_CHALLENGE* cell and feed those to our AES encryption in CBC mode. We use the output of the HKDF as our temporary private NTOR key and use the *KEY_SEED* as our randomness by feeding it again into our AES cipher and using the output. We can see an overview of the circuit creation in Figure 4.3. Since routers only receive hashes of our ciphertext, it is impossible to draw any conclusion about the current state of our AES encryption. Also, since every preceding encryption influences succeeding encryption, it will be impossible to predict the ciphertext without knowing every exchanged secret, even if one knows the encryption key. For our randomness, we only use the output of the AES encryption and never the input values. So even if a malicious router generates a bad *KEY_SEED*, if we encrypted enough random inputs previously, the output is still secure.

```
function GetRandomBytes()
    c_ID ← rand(time)
    AUTH_CHALLENGE ← CreateCircuit(c_ID)
    t ← AES(AUTH_CHALLENGE)
    x ← HKDF(t, ":NTORKey")
    X ← GeneratePublicX25519Key(x)
    KEY_SEED ← NTORHandshake(x, X, ID, Y)
    return AES(KEY_SEED)
end function
```

Algorithm 4.3 Get random string from router

```
function Expand(x)
    random ← empty
    y ← GetRandomBytes()
    for i ← 1 to number of expansion do
        x ← AES(y)
        y ← HKDF(x, ":RandomExtract")
        Append y to random
    end for
    return random
end function
```

Algorithm 4.4 Expand randomness

4.5.5 Randomness expansion

In the case that one needs more random data, we implemented a randomness expansion. Using this, we collect 32 bytes of randomness and encrypt and hash them multiple times to generate a pseudorandom sequence. For many use cases, this is enough. A kernel's internal random number generator works like this.

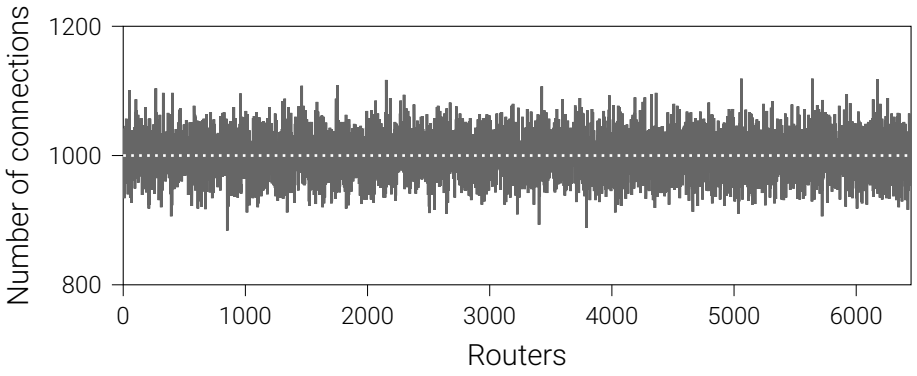


Figure 4.4 First router selection while bootstrapping

4.6 Evaluation

We analyze the TORC and its randomness collection. We evaluate previous claims and analyze the performance, overhead, quality of randomness, and security. First, we want to verify our claim that TORC equally selects TOR routers.

4.6.1 Router selection

We ran 6.500.000 initial setups for the initial router selection using a timestamp as a seed. With 6500 routers, TORC should connect about 1000 times to each router. We can see Figure 4.4 that the distributed connections, and that no single router gets too many requests.

4.6.2 Quality of randomness

We used a popular software test package ENT [105] to apply statistical tests on the generated sequences. ENT provides a comprehensive analysis of randomness testing for cryptographic applications. It performs statistical tests over an input sequence and produces output according to common randomness properties. For each test, the ENT suite program generates entropy, χ^2 value, arithmetic mean value μ, Monte Carlo value for π, and serial correlation coefficient. These tests measure different properties of the strings appearing in the tested sequence.

We apply these tests on sequences of strings collected from (1) TORC without expansion, (2) TORC with expansion, (3) the C rand() using a recent GCC 9.2.0 compiler, and (4) the /dev/random device of a current macOS 10.14.6. For each of these

	TORC	Alexa	rand()	/dev/random
Entropy (bits/byte)	7.9998	7.9998	7.9546	7.9998
Compression (%)	0	0	0	0
χ^2	243.75	278.10	67367.89	248.40
exceeds (%)	68.31	15.32	0.01	60.46
μ	127.4	127.5	111.63	127.50
π	3.146	3.143	3.482	3.141
Serial correlation	0.001	0.0009	-0.048	-0.001

Table 4.5 Statistical tests over sequences produced by TORC, Alexa Top 1M, the C rand() function and the /dev/random device of macOS

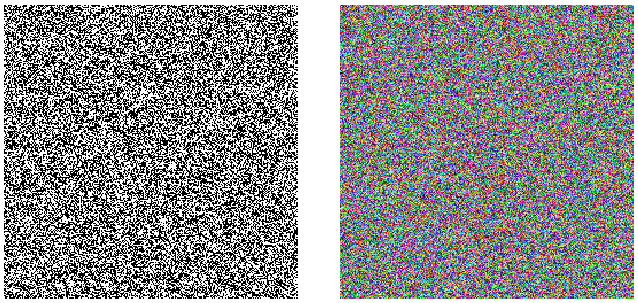


Figure 4.5 Visualization of collected randomness using TORC as black and white bits (left) and RGB values (right)

sources, we collect a 1MB file of random bits. We can see the results in Table 4.5. The expansion mechanism in TORC uses 32 randomly collected bytes and encrypts and hashed these multiple times to generate longer pseudorandom sequences. In this case, 32 bytes will generate a sequence of 32768 bytes. Pseudorandom number sequences are still very secure if seeded and reseeded well.

The abbreviation *exceeds %* of χ^2 in the table describes the degree to which the test suspects that the sequence is not random. If the percentage is above 99%, we consider it as not random. We refer an interested reader to Ruhkin et al. [138] for further information about the statistical tests.

Yfantis et al. [165] suggested visual and acoustic tests for random number generators. We show the visualization of 65536 random bits collected with TORC in Figure 4.5. A black pixel represents a binary 0, and a white pixel represents a binary 1. There should be no visible patterns. We also included a version where we use three bytes as RGB values to visualize 196608 bytes of randomness collected using TORC.

Method	Time
rand()	0.02s
/dev/random	0.04s
TORC (1024 expansion)	21.35s
TORC (no expansion)	22534.40s

Table 4.6 Time to collect 1MB (1,048,576 bytes) of random data

Latency	Time (cached)	Time (uncached)
0ms	56ms	83ms
1ms	80ms	118ms
2ms	104ms	153ms
5ms	176ms	258ms
10ms	296ms	433ms
20ms	536ms	783ms
50ms	1256ms	1833ms
100ms	2456ms	3583ms
200ms	4856ms	7083ms
500ms	12056ms	17583ms

Table 4.7 Time to collect 32 Bytes of random data using different latencies and a fixed bandwidth of 1 MBit/s

4.6.3 Performance

Randomness collection through internet connections can never compete with hardware implementation or pure user space tools. However, we want to provide an overview of how TORC performs in terms of speed. Table 4.6 shows how TORC performs against the C rand() function and the /dev/random device. We used a 1 GBit/s connection for this test. Two factors are slowing down the collection of randomness: (1) unresponsive or failing routers, which take up time without generating any randomness, and (2) latency. The bandwidth almost does not affect the performance. TOR cells are 514 bytes, and we spend most of the time waiting for responses. Latency, however, has an enormous influence on speed.

Table 4.7 shows a simulation using different latencies. Here we assume a bandwidth of 1 MBit/s since this is more than needed, and most devices connected to the Internet usually have 1 MBit/s or more. However, we do not include the failing

Key size	Time (50ms)	Time (100ms)
256 bit	1256ms	2456ms
512 bit	2512ms	4912ms
1024 bit	5024ms	9824ms
2048 bit	10048ms	19648ms
4096 bit	20096ms	39296ms

Table 4.8 Time to collect different key sizes using a fixed bandwidth of 1 MBit/s

routers in the simulation. When we connect to a router, we must fetch its descriptor and NTOR keys. To speed up the randomness collection, we cache these. If we connect to a router again, we can use the cached values. We included values where we assume everything is cached and where nothing is cached. One should consider the simulation a rough estimation since the actual latency depends on the routing of the internet provider. We also did not include the processing of the information by the device. Devices can range from small ones, like a Raspberry Pi, to high-end computers or Mainframes. The calculation speed does not matter for a faster computer since it is almost instant. It does, however, for smaller devices like, e.g., IoT clients.

In Table 4.8, we can see what it would mean for the generation of different key sizes. Without a cached descriptor and high latency, it can take 39 seconds to gather 4096 bits of randomness. A common use-case for this amount of random bits would be the generation of an RSA key.

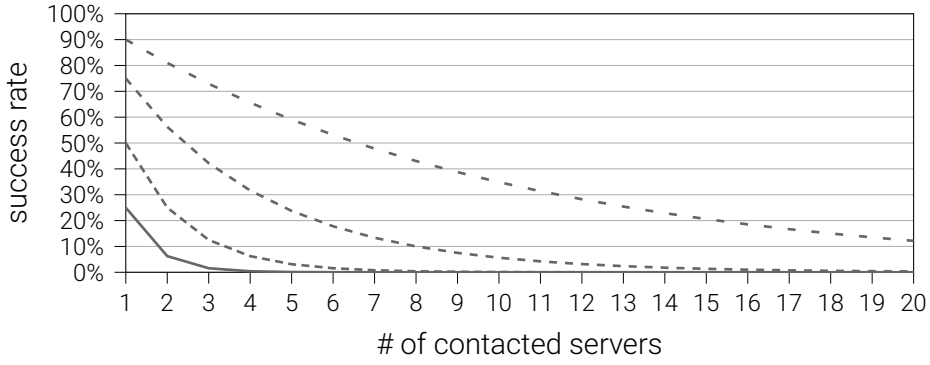


Figure 4.6 Simulation of the success rate of an attacker

4.6.4 Security analysis

Using n servers ensures security even if some, but not all, are compromised, provide vulnerable randomness, or if the attacker can eavesdrop on some of the networks or paths. We assume that an attacker knows the initial seed, e.g., if it is just a timestamp. When using an AES CBC cipher, an attacker loses if we successfully connect to a non-malicious server. DPRG is also secure against past and future break-ins. Compromising the system does not break previous or future connections. The AES CBC cipher continuously permutates, and old states get discarded.

4.6.4.1 At least one good server

The probability when selecting n servers that at least one provides good pseudo-randomness is high. Denote the set of all potential servers with S, and C is the set corrupted servers. First, the attacker selects the set C of corrupted servers, then DPRG selects n servers out of S. We can describe the success probability of an attack as

$$\Pr(\text{attack}) = \prod_{i=1}^{n} \frac{|C|}{|S|}.$$

We can see a visual representation with 95%, 90%, 75% 50%, and 25% corrupted servers in Figure 4.6. With enough initial rounds, even strong attackers become weak.

4.6.4.2 Hijacking attackers

Since a MitM attacker cannot subvert the security of a DPRG, it can attempt to perform launch Border Gateway Protocol (BGP) prefix hijacks. BGP is known to be vulnerable to prefix hijack attacks [1,2,18,31]. In prefix hijacks, the attacker hijacks all the traffic of a victim network.

We evaluate the attacker's ability to exploit the insecurity of BGP to hijack traffic between the networks operating a DPRG and the servers. In this case, both the victim and the attacker announce the victim's BGP prefix. The probability that the attacker attracts more than 90% of the web servers is 2%. The simulation also shows that attackers that can hijack traffic from 90% of the web servers would disconnect the victim from the rest of the Internet. Only 0.20% of the attackers can successfully launch the attack while maintaining their route to the victim to relay packets between the victim and the rest of the Internet to avoid detection.

To prevent a DPRG from generating randomness, the attacker must hijack traffic from almost all the servers. The consequences would be a disruption of all the ASes on the Internet.

4.6.4.3 Reversing HKDF

We showed that every router receives a HKDF value of our cipher block. The 256-bit input space, even when using fixed keys, makes it impossible to brute force or generate rainbow tables for a given output. It would require $2^{256} \cdot 32$ bytes, which is a 79 decimal digit number, to store a 256-bit rainbow table.

4.6.4.4 Untrusted local ISP

If we trust the local ISP, we can securely collect randomness since not all paths overlap. There will be at least one non-malicious path. However, if the ISP is untrusted, there are two further requirements. The services must provide a public-key infrastructure (PKI) to authenticate the servers. We achieved this by using the TOR consensus. But if we were using SSH servers for our DPRG implementation, we would have no PKI. The ISP can monitor all traffic. We need one initial seed that is unknown to the ISP. When a DPRG finishes its initialization using the provided seed, it can refresh it. The next time it starts, there is no need for a new one. We can also use existing DPRG instances to generate new ones. E.g., we can use TORC to collect a random seed which we can use.

4.7 Conclusion

Randomness is essential for bootstrapping cryptography. Often, randomness is difficult to obtain or generate in practice, and we have seen many high-profile PRG failures and vulnerabilities over the years.

We proposed DPRG, a novel approach to the pseudo-randomness generation problem. We have shown how to collect randomness using public services on the Internet. We explored different public services which a DPRG could use. We highlighted our implementation using TOR called TORC and evaluated its quality, performance, and security.

DPRG can revolutionize not only the current state of randomness generation but can push forward support and usage of crypto in restricted devices, e.g., that do not have access to sources of entropy, such as IoT devices or smart cards. It is also a worthy alternative to hardware RNGs on virtualized environments or any device where the RNG is unreliable. One advantage of DPRG is the independence of the hardware entropy. It runs only in software mode and, besides trivial tasks like opening network sockets, needs no help from the operating system.

CHAPTER 5
Summary

More people use the Internet. Their safety relies on the cryptographic backbones of the Internet. Without cryptography, online transactions, login credentials, and more would be insecure. We must maintain the security, even if new attacks appear. E.g., when post-quantum computers start to threaten current cryptographic algorithms, we must replace them with new ones that are secure against post-quantum computers.

In this thesis, we look for the answer to the question. "What does it take to bootstrap cryptography on the Internet, and how can we improve it?" We answer the question by evaluating three of the five requirements we defined. We show how to improve transport layers, PKIs, and randomness collection.

In Chapter 2, we investigate the effect of transport layers on cryptographic algorithms, which is a direction that no one previously explored. We show that there are no transport layers that replace all others. We introduce a framework for secure computation applications which can utilize different transport layers depending on the application.

In Chapter 3, we evaluate the robustness of the PKI of the Internet. We show the world's first attack against domain validation utilizing off-path DNS cache poisoning. We present an optimized BGP simulator for evaluating Internet BGP paths. We explain how we optimized the simulator and how we can determine valid BGP paths on the Internet. We also analyze how vulnerable name servers on the Internet are. We analyze attack probabilities using same-prefix and sub-prefix hijacks. Finally, we present a countermeasure using distributed domain validation.

In Chapter 4, we present a novel approach to collect randomness on the Internet. We evaluate the feasibility of a distributed pseudo-random generator and introduce a reference implementation using the TOR network. We analyze the performance and compare it to existing PRNGs. We measure the overhead involved in this method and the quality of randomness.

We demonstrate the importance of requirements to cryptography and show problems that currently exist. We show that public-key infrastructures are a fragile construct and need improvements. We further show how choosing a suitable transport layer allows cryptographic applications that would otherwise be completely unusable. We also demonstrate that we can collect cryptographically secure randomness even on devices that lack entropy. Overall, we show cryptography from a different perspective. Instead of mathematical formulas and proofs, we focus on the practical application of cryptography.

CHAPTER 6
Future Work

Research is ongoing work. However, our time is finite, and we must concentrate on chosen directions and cannot follow them all. Also, to finish this thesis, we had to stop research at a point. These two arguments create space for future work.

We show a transport layer framework for secure computation. From here, there are so many directions we can go. We can investigate other transport layers and cryptographic protocols. We can look at Secure Computation with more than two parties.

Or we can go in a completely different direction, like developing application-aware transport layers. Here, applications could inform transport layers about the requirements. E.g., transport layers could switch from a file transfer mode to a loss-resistant burst mode for quick handshakes.

For the public-key infrastructure of the Internet, there is a lot of future work to do. For one, we must deploy cryptographic extensions like DNSSEC. The backbone of the Internet, i.e., DNS and BGP, have many flaws. A good research direction would be the evaluation of new DNS and BGP alternatives that already utilize the distributed nature of the Internet for security. However, these changes would be massive. We should not expect anything more than theoretical research soon.

The simulation of BGPs paths is a convenient utility. The Internet is a live system, and we cannot take parts of it offline for evaluation purposes. So, the tool we depend on is simulation. Of course, further improvement of our simulator can be a research direction. Another future work would be to build a complex simulation framework

build on top of our simulator. With it, even researchers without programming skills should be able to perform simulations.

We demonstrated the distributed domain validation. While our use case was the web's PKI, one could utilize distributed agents to validate different services. Another research topic could be the development of an open and decentralized validation infrastructure. Many participants can access other agents by giving them access to their agents. Further evaluation of a fully deployed DDV setup may also be another direction that future works may follow.

Network randomness collectors are a new research direction. Not much research exists for these. Future works could include the evaluation of low-power devices. One should evaluate the feasibility of network randomness collection on IoT devices. These devices come with limited resources, and we need some optimizations for these devices.

Also, we should integrate the randomness collection into existing libraries. E.g., OpenSSL and other libraries perform the SSL connections on most operating systems. We could extract randomness from, e.g., handshakes and seed the internal randomness.

Of course, the two requirements we did not cover also require research. While performance is still one of the main goals besides security, usability can be difficult. The easier it is to use algorithms, the more users will use them. However, under no circumstance should we sacrifice security for usability. Security must be the number one requirement.

Overall, we consider all of our implementations and mechanism as a starting point. We hope that future works will extend these in many directions.

APPENDIX A

Appendix

ASN	Name	IPs
16625	AKAMAI-AS, US	11.51
16509	AMAZON-02, US	7.39
7018	ATT-INTERNET4, US	4.35
14618	AMAZON-AES, US	2.88
20940	AKAMAI-ASN1, US	2.35
8075	MICROSOFT-CORP-MSN-AS-BLOCK, US	1.74
15169	GOOGLE, US	1.61
13335	CLOUDFLARENET, US	1.58
209	CENTURYLINK-US-LEGACY-QWEST, US	1.53
16276	OVH, FR	1.39

Table A.1 Top 10 ASN distribution for HTTPS

ASN	Name	IPs
133612	VODAFONE-AS-AP Vodafone Australia Pty Ltd, AU	5.27
16276	OVH, FR	4.60
46606	UNIFIEDLAYER-AS-1, US	3.66
15169	GOOGLE, US	3.55
12824	HOMEPL-AS, PL	2.18
26658	HENGTONG-IDC-LLC, US	2.07
24940	HETZNER-AS, DE	1.79
27680	TELEFONICA MOVIL DE CHILE S.A., CL	1.31
14061	DIGITALOCEAN-ASN, US	1.21
6325	ILLINOIS-CENTURY, US	1.05

Table A.2 Top 10 ASN distribution for SMTP(S)

ASN	Name	IPs
16509	AMAZON-02, US	7.49
37963	CNNIC-ALIBABA-CN-NET-AP Hangzhou Alibaba Advertising Co.,Ltd., CN	4.49
14061	DIGITALOCEAN-ASN, US	4.48
15169	GOOGLE, US	4.19
14618	AMAZON-AES, US	3.23
16276	OVH, FR	3.17
133612	VODAFONE-AS-AP Vodafone Australia Pty Ltd, AU	1.66
24940	HETZNER-AS, DE	1.62
4134	CHINANET-BACKBONE No.31,Jin-rong Street, CN	1.52
45090	CNNIC-TENCENT-NET-AP Shenzhen Tencent Computer Systems Company Limited, CN	1.33

Table A.3 Top 10 ASN distribution for SSH

Bibliography

[1] *Hijack Event Today by Indosat*, http://www.bgpmon.net/hijack-event-today-by-indosat (unpublished).

[2] *New Threat: Targeted Internet Traffic Misdirection*, http://www.renesys.com/2013/11/mitm-internet-hijacking (unpublished).

[3] ars technica, *How a months-old AMD microcode bug destroyed my weekend*, https://arstechnica.com/gadgets/2019/10/how-a-months-old-amd-microcode-bug-destroyed-my-weekend/ (2019).

[4] M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, Global Authentication in an Untrustworthy World., In HotOS (2013).

[5] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J.A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta *et al.*, Imperfect forward secrecy: How Diffie-Hellman fails in practice, In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM (2015).

[6] Alexa, *Top Sites*, https://www.alexa.com/topsites (unpublished).

[7] M. Alicherry and A.D. Keromytis, Doublecheck: Multi-path verification against man-in-the-middle attacks, In 2009 IEEE Symposium on Computers and Communications, IEEE (2009).

[8] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, Data center tcp (dctcp), *ACM SIGCOMM computer communication review 41*(4), 63–74 (2011).

[9] B. Amann, M. Vallentin, S. Hall, and R. Sommer, Extracting certificates from live traffic: A near real-time SSL notary service, *Technical Report TR-12-014* (2012).

[10] D. Anderson, Splinternet Behind the Great Firewall of China, *Queue 10*(11), 40 (2012).

[11] APNIC, *DNSSEC Validation Rate by country*, https://stats.labs.apnic.net/dnssec (2021).

[12] D.W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J.I. Pagter, N.P. Smart, and R.N. Wright, From Keys to Databases - Real-World Applications of Secure Multi-Party Computation, *Comput. J. 61*(12), 1749–1771 (2018).

[13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS Security Introduction and Requirements*, RFC, no. 4033 (RFC Editor, 2005). (http://www.rfc-editor.org/rfc/rfc4033.txt)

[14] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *Protocol Modifications for the DNS Security Extensions*, RFC, no. 4035 (RFC Editor, 2005). (http://www.rfc-editor.org/rfc/rfc4035.txt)

[15] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *Resource Records for the DNS Security Extensions*, RFC, no. 4034 (RFC Editor, 2005). (http://www.rfc-editor.org/rfc/rfc4034.txt)

[16] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, More efficient oblivious transfer and extensions for faster secure computation, In ACM Conference on Computer and Communications Security (ACM, 2013).

[17] Y. Aumann and Y. Lindell, Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries, In S.P. Vadhan (Ed.) Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings (Springer, 2007).

[18] H. Ballani, P. Francis, and X. Zhang, A study of prefix hijacking and interception in the Internet, *ACM SIGCOMM Computer Communication Review 37*(4), 265–276 (2007).

[19] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for Weak Participants, In CCS (ACM, 2018).

[20] E. Barker and A. Roginsky, *Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths*, NIST Special Publication (2011).

[21] D. Basin, C. Cremers, T. Hyuni-jin, A. Perrig, R. Sasse, and P. Szalachowski, Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure, *IEEE Transactions on Dependable and Secure Computing* (2016).

[22] D. Beaver, S. Micali, and P. Rogaway, The Round Complexity of Secure Protocols (Extended Abstract), In STOC (ACM, 1990).

[23] M. Bellare, V.T. Hoang, S. Keelveedhi, and P. Rogaway, Efficient Garbling from a Fixed-Key Blockcipher, In IEEE Symposium on Security and Privacy (IEEE Computer Society, 2013).

[24] M. Bellare, V.T. Hoang, and P. Rogaway, Foundations of garbled circuits, In ACM Conference on Computer and Communications Security (ACM, 2012).

[25] M. Bellare and P. Rogaway, Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols (Association for Computing Machinery, New York, NY, USA, 1993).

[26] D.J. Bernstein, *[Cfrg] 25510 naming*, https://mailarchive.ietf.org/arch/msg/cfrg/-9LEdnzVrE5RORux3Oo_oDDRksU (unpublished).

[27] D.J. Bernstein, Curve25519: New Diffie-Hellman Speed Records., In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin (Eds.) Public Key Cryptography (Springer, 2006).

[28] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, Bamboozling Certificate Authorities with BGP, In W. Enck and A.P. Felt (Eds.) 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018 (USENIX Association, 2018).

[29] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, Using BGP to acquire bogus TLS certificates, *HotPETS'17* (2017).

[30] M.S. Blog, *DigiNotar Removal Follow Up*, https://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/ (2011).

[31] R. Blog, *Renesys Blog - Pakistan Hijacks YouTube*, http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml (2008).

[32] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudorandom bits, *SIAM journal on Computing 13*(4), 850–864 (1984).

[33] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht, How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation, In Financial Cryptography (Springer, 2015).

[34] P. Bogetoft, D.L. Christensen, I. Damgård, M. Geisler, T.P. Jakobsen, M. Krøigaard, J.D. Nielsen, J.B. Nielsen, K. Nielsen, J. Pagter *et al.*, Secure Multiparty Computation Goes Live, In Financial Cryptography (Springer, 2009).

[35] Z. Brakerski, S. Goldwasser, G.N. Rothblum, and V. Vaikuntanathan, Weak verifiable random functions, In Theory of Cryptography Conference, Springer (2009).

[36] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, Domain Validation++ For MitM-Resilient PKI, In D. Lie, M. Mannan, M. Backes, and X. Wang (Eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018 (ACM, 2018).

[37] M. Brandt, T. Dai, H. Shulman, and M. Waidner, Evaluating Resilience of Domains in PKI, In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Association for Computing Machinery, 2021).

[38] M. Brandt, C. Orlandi, K. Shrishak, and H. Shulman, Transputation: Transport Framework for Secure Computation, In F. Kiefer and D. Loebenberger (Eds.) Crypto day matters 30 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[39] M. Brandt, C. Orlandi, K. Shrishak, and H. Shulman, Optimal Transport Layer for Secure Computation, In Proceedings of the 17th International Joint Conference on e-Business and Telecommunications, Volume 2: SECRYPT, Lieusaint, Paris, France, July 8-10, 2020 (ICETE, 2020).

[40] M. Brandt and H. Shulman, Optimized BGP Simulator for Evaluation of Internet Hijacks, In 40th IEEE Conference on Computer Communications, INFOCOM 2021, Virtual Conference, Mai 11-13, 2021 (IEEE, 2021).

[41] M. Brandt, H. Shulman, and M. Waidner, Internet As a Source of Randomness, In Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets 2018, Redmond, WA, USA, November 15-16, 2018 (ACM, 2018).

[42] M. Brandt, H. Shulman, and M. Waidner, Distributed Domain Validation (DDV), In M. Selhorst, D. Loebenberger, and M. Nüsken (Eds.) Crypto day matters 31 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[43] M. Brandt, H. Shulman, and M. Waidner, Internet As a Source of Randomness, In F. Kiefer and D. Loebenberger (Eds.) Crypto day matters 30 (Gesellschaft für Informatik e.V. / FG KRYPTO, Bonn, 2019).

[44] Bruce Schneier, *Random Number Bug in Debian Linux*, https://www.schneier.com/blog/archives/2008/05/random_number_b.html (2008).

[45] C. Caini and R. Firrincieli, TCP Hybla: a TCP enhancement for heterogeneous networks, *Int. J. Satellite Communications Networking 22*(5), 547–566 (2004).

[46] N. Cardwell, Y. Cheng, C.S. Gunn, S.H. Yeganeh, and V. Jacobson, BBR: Congestion-Based Congestion Control, *ACM Queue 14*(5), 20–53 (2016).

[47] CAIDA, *Inferred AS Relationships Dataset*, https://www.caida.org/data/as-relationships (unpublished).

[48] V. Cheval, M. Ryan, and J. Yu, DTKI: a new formalized PKI with no trusted parties, *arXiv preprint arXiv:1408.1023* (2014).

[49] S.G. Choi, J. Katz, R. Kumaresan, and H.S. Zhou, On the Security of the "Free-XOR" Technique, In TCC (Springer, 2012).

[50] B. Chor and O. Goldreich, Unbiased bits from sources of weak randomness and probabilistic communication complexity, *SIAM Journal on Computing 17*(2), 230–261 (1988).

[51] C.S. Chow and A. Herzberg, Network Randomization Protocol: A Proactive Pseudo-Random Generator, In Proceedings of the 5th Symposium on UNIX Security (USENIX Association, Berkeley, CA, USA, 1995).

[52] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, Planetlab: an overlay testbed for broad-coverage services, *ACM SIGCOMM Computer Communication Review 33*(3), 3–12 (2003).

[53] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B.M. Maggs, A. Mislove, and C. Wilson, A longitudinal, end-to-end view of the DNSSEC ecosystem, In USENIX Security (2017).

[54] Cloudflare, Inc., *Randomness 101: LavaRand in Production*, https://blog.cloudflare.com/randomness-101-lavarand-in-production/ (2017).

[55] A. Cohen, Y. Gilad, A. Herzberg, and M. Schapira, Jumpstarting BGP Security with Path-End Validation, In M.P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti (Eds.) Proceedings of the ACM SIGCOMM 2016 Conference, Florianopolis, Brazil, August 22-26, 2016 (ACM, 2016).

[56] R. Colbeck and R. Renner, Free randomness can be amplified, *Nature Physics 8*(6), 450 (2012).

[57] H. Corrigan-Gibbs and S. Jana, Recommendations for Randomness in the Operating System, or How to Keep Evil Children out of Your Pool and Other Random Facts., In HotOS (2015).

[58] T. Dai, H. Shulman, and M. Waidner, DNSSEC Misconfigurations in Popular Domains, In International Conference on Cryptology and Network Security, Springer (2016).

[59] T. Dai, H. Shulman, and M. Waidner, Let's Downgrade Let's Encrypt, In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS 2021, Virtual Conference, November 15-19, 2021 (ACM, 2021).

[60] L. Daigle, *WHOIS Protocol Specification*, RFC, no. 3912 (RFC Editor, 2004).

[61] J.P. Degabriele, K.G. Paterson, J.C. Schuldt, and J. Woodage, Backdoors in pseudorandom number generators: Possibility and impossibility results, In Annual International Cryptology Conference, Springer (2016).

[62] D. Demmler, T. Schneider, and M. Zohner, ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation, In NDSS (The Internet Society, 2015). (Code: https://github.com/encryptogroup/ABY)

[63] Deploy260, *Email Hijacking*, https://www.internetsociety.org/blog/2014/09/email-hijacking-new-research-shows-why-we-need-dnssec-now/ (2014).

[64] R. Dingledine, *Pre-alpha: run an onion proxy now!*, https://archives.seul.org/or/dev/Sep-2002/msg00019.html (unpublished).

[65] R. Dingledine, N. Mathewson, and P. Syverson, Tor: The Second-generation Onion Router, In Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13 (USENIX Association, Berkeley, CA, USA, 2004).

[66] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs, Security analysis of pseudo-random number generators with input: /dev/random is not robust, In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM (2013).

[67] Donald E. Eastlake 3rd and C.W. Kaufman, *Domain Name System Security Extensions*, RFC, no. 2065 (RFC Editor, 1997). (http://www.rfc-editor.org/rfc/rfc2065.txt)

[68] L. Dorrendorf, Z. Gutterman, and B. Pinkas, Cryptanalysis of the windows random number generator, In Proceedings of the 14th ACM conference on Computer and communications security, ACM (2007).

[69] V. Dukhovni and W. Hardaker, *The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance*, RFC, no. 7671 (RFC Editor, 2015).

[70] Z. Durumeric, E. Wustrow, and J.A. Halderman, ZMap: Fast Internet-wide Scanning and Its Security Applications., In USENIX Security Symposium (2013).

[71] P. Eckersley and J. Burns, *An observatory for the SSLiverse*, DEFCON'18 (2010).

[72] P. Eckersley, Sovereign keys: A proposal to make https and email more secure, *Electronic Frontier Foundation 18* (2011).

[73] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, SCAPI: The Secure Computation Application Programming Interface, *IACR Cryptology ePrint Archive 2012* 629 (2012). (Code: https://github.com/cryptobiu/libscapi)

[74] R. Elz and R. Bush, *Clarifications to the DNS Specification*, RFC, no. 2181 (RFC Editor, 1997).

[75] A. Everspaugh, Y. Zhai, R. Jellinek, T. Ristenpart, and M. Swift, Not-so-random numbers in virtualized Linux and the Whirlwind RNG, In Security and Privacy (SP), 2014 IEEE Symposium on, IEEE (2014).

[76] T. Garfinkel and M. Rosenblum, When Virtual Is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments., In HotOS (2005).

[77] Y. Gilad and A. Herzberg, Fragmentation Considered Vulnerable, *ACM Transactions on Information and System Security (TISSEC) 15*(4), 16:1–16:31 (2013). (A preliminary version appeared in WOOT 2011.)

[78] P. Gill, M. Schapira, and S. Goldberg, Modeling on quicksand: dealing with the scarcity of ground truth in interdomain routing data, *Comput. Commun. Rev. 42*(1), 40–46 (2012).

[79] I. Goldberg and D. Wagner, Randomness and the Netscape browser, *Dr Dobb's Journal-Software Tools for the Professional Programmer 21*(1), 66–71 (1996).

[80] O. Goldreich, S. Micali, and A. Wigderson, How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority, In STOC (ACM, 1987).

[81] Google, *Certificate Transparency*, https://chromium.googlesource.com/chromium/src/+/refs/heads/main/net/docs/certificate-transparency.md (2018).

[82] Y. Gu and R.L. Grossman, UDTv4: Improvements in Performance and Usability, In GridNets (Springer, 2008).

[83] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas, Fast Garbling of Circuits Under Standard Assumptions, In ACM Conference on Computer and Communications Security (ACM, 2015).

[84] P. Gutmann, Software generation of random numbers for cryptographic purposes, In Proceedings of the 1998 Usenix Security Symposium (1998).

[85] Z. Gutterman, B. Pinkas, and T. Reinman, Analysis of the linux random number generator, In Security and Privacy, 2006 IEEE Symposium on, IEEE (2006).

[86] S. Ha, I. Rhee, and L. Xu, CUBIC: a new TCP-friendly high-speed TCP variant, *Operating Systems Review 42*(5), 64–74 (2008).

[87] T.J. Hacker, B.D. Athey, and B. Noble, The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network, In IPDPS (IEEE Computer Society, 2002).

[88] S. Halevi, Advanced Cryptography: Promise and Challenges, In ACM Conference on Computer and Communications Security (ACM, 2018).

[89] P. Hallam-Baker and R. Stradling, *DNS Certification Authority Authorization (CAA) Resource Record*, RFC, no. 6844 (RFC Editor, 2013).

[90] S. Hao, Y. Zhang, H. Wang, and A. Stavrou, End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks, In 27th USENIX Security Symposium (USENIX Security 18), USENIX Association (2018).

[91] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, SoK: General Purpose Compilers for Secure Multi-Party Computation, In 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019 (IEEE, 2019).

[92] B. Hemenway, S. Lu, R. Ostrovsky, and W.W. IV, High-Precision Secure Computation of Satellite Collision Probabilities, In SCN (Springer, 2016).

[93] N. Heninger, Z. Durumeric, E. Wustrow, and J.A. Halderman, Mining your Ps and Qs: Detection of widespread weak keys in network devices, In Presented as part of the 21st USENIX Security Symposium (USENIX Security 12) (2012).

[94] A. Herzberg and H. Shulman, Security of Patched DNS, In Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings (2012).

[95] A. Herzberg and H. Shulman, Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org, In Communications and Network Security (CNS), 2013 IEEE Conference on, IEEE (2013).

[96] A. Herzberg and H. Shulman, Vulnerable Delegation of DNS Resolution, In Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings (2013).

[97] A. Herzberg and H. Shulman, Socket Overloading for Fun and Cache Poisoning, In C.N.P. Jr. (Ed.) ACM Annual Computer Security Applications Conference (ACM ACSAC), New Orleans, Louisiana, U.S. (2013).

[98] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, 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, ACM (2011).

[99] M. Hu, Taxonomy of the Snowden Disclosures, *Wash & Lee L. Rev. 72* 1679–1989 (2015).

[100] Y. Huang, D. Evans, and J. Katz, Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?, In NDSS (The Internet Society, 2012).

[101] Y. Huang, D. Evans, J. Katz, and L. Malka, Faster Secure Two-Party Computation Using Garbled Circuits, In USENIX Security Symposium (USENIX Association, 2011).

[102] A. Hubert and R. van Mook, *Measures for Making DNS More Resilient against Forged Answers*, RFC, no. 5452 (RFC Editor, 2009).

[103] V.C. Inc., *Monthly IP Latency Data - Verizon Enterprise Solutions*, https://www.verizon.com/business/terms/latency/ (2019).

[104] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, Extending Oblivious Transfers Efficiently, In CRYPTO (Springer, 2003).

[105] John Walker, *ENT: A Pseudorandom Number Sequence Test Program*, http://www.fourmilab.ch/random/ (2008).

[106] John Walker, *HotBits: Genuine random numbers, generated by radioactive decay*, https://www.fourmilab.ch/hotbits/ (2017).

[107] D. Kaminsky, It's the End of the Cache As We Know It, In Black Hat conference (2008). (http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf)

[108] B. Kerrigan and Y. Chen, A study of entropy sources in cloud computers: random number generation on cloud hosts, *Computer Network Security* 286–298 (2012).

[109] T.H.J. Kim, L.S. Huang, A. Perring, C. Jackson, and V. Gligor, Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure, In Proceedings of the 22nd international conference on World Wide Web, ACM (2013).

[110] A. Klein, H. Shulman, and M. Waidner, Counting in the Dark: Caches Discovery and Enumeration in the Internet, In The 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (2017).

[111] A. Klein, H. Shulman, and M. Waidner, Internet-Wide Study of DNS Cache Injections, In INFOCOM (2017).

[112] J. Knockel and J.R. Crandall, Counting Packets Sent Between Arbitrary Internet Hosts., In FOCI (2014).

[113] V. Kolesnikov and T. Schneider, Improved Garbled Circuit: Free XOR Gates and Applications, In ICALP (2) (Springer, 2008).

[114] H. Krawczyk and P. Eronen, *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*, RFC, no. 5869 (RFC Editor, 2010).

[115] B. Kreuter, Secure multiparty computation at Google, In Real World Crypto Conference (RWC) (2017).

[116] mapsofworld.com, *Top Ten Countries with Highest number of PCs*, https://www.mapsofworld.com/world-top-ten/world-top-ten-personal-computers-users-map.html (unpublished).

[117] B. Laurie, A. Langley, and E. Kasper, *Certificate transparency*, Technical report (2013).

[118] LavaRnd.org, *LavaRnd Number Generator*, http://www.lavarnd.org/lavarnd.html (2000).

[119] C. Liu, X.S. Wang, K. Nayak, Y. Huang, and E. Shi, ObliVM: A Programming Framework for Secure Computation, In IEEE Symposium on Security and Privacy (IEEE Computer Society, 2015).

[120] S. Liu, T. Basar, and R. Srikant, TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks, *Perform. Eval. 65*(6-7), 417–440 (2008).

[121] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, Fairplay - Secure Two-Party Computation System, In USENIX Security Symposium (USENIX, 2004).

[122] L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, and R. Wheeler, *A Method for Transmitting PPP Over Ethernet (PPPoE)*, RFC, no. 2516 (RFC Editor, 1999). (http://www.rfc-editor.org/rfc/rfc2516.txt)

[123] P. Mockapetris, *Domain names - implementation and specification*, STD, no. 13 (RFC Editor, 1987). (http://www.rfc-editor.org/rfc/rfc1035.txt)

[124] J. Mogul and S. Deering, *Path MTU discovery*, RFC, no. 1191 (RFC Editor, 1990). (http://www.rfc-editor.org/rfc/rfc1191.txt)

[125] J. Nagle, Congestion control in IP/TCP internetworks, *RFC 896* (1984).

[126] M. Naor and B. Pinkas, Efficient oblivious transfer protocols, In SODA (ACM/SIAM, 2001).

[127] M. Naor, B. Pinkas, and R. Sumner, Privacy preserving auctions and mechanism design, In EC (1999).

[128] Netcraft, *SSL Survey*, https://www.netcraft.com/internet-data-mining/ssl-survey/ (2018).

[129] J.B. Nielsen, T. Schneider, and R. Trifiletti, Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO, In NDSS (The Internet Society, 2017).

[130] H. Obata, K. Tamehiro, and K. Ishida, Experimental evaluation of TCP-STAR for satellite Internet over WINDS, In 2011 Tenth International Symposium on Autonomous Decentralized Systems (ISADS), IEEE (2011).

[131] K. Park, V.S. Pai, L.L. Peterson, and Z. Wang, CoDNS: Improving DNS Performance and Reliability via Cooperative Lookups., In OSDI (2004).

[132] B. Pinkas, T. Schneider, N.P. Smart, and S.C. Williams, Secure Two-Party Computation Is Practical, In ASIACRYPT (Springer, 2009).

[133] B. Pinkas, T. Schneider, and M. Zohner, Scalable Private Set Intersection Based on OT Extension, *ACM Trans. Priv. Secur. 21*(2), 7:1–7:35 (2018).

[134] L. Poole and V.S. Pai, ConfiDNS: Leveraging Scale and History to Improve DNS Security., In WORLDS (2006).

[135] J. Postel, *User Datagram Protocol*, STD, no. 6 (RFC Editor, 1980). (http://www.rfc-editor.org/rfc/rfc768.txt)

[136] J. Postel, *Internet Protocol*, STD, no. 5 (RFC Editor, 1981). (http://www.rfc-editor.org/rfc/rfc791.txt)

[137] T. Ristenpart and S. Yilek, When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography., In NDSS (2010).

[138] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, Technical report (Booz-Allen and Hamilton Inc Mclean Va, 2001).

[139] T. Schneider and M. Zohner, GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits, In Financial Cryptography (Springer, 2013).

[140] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, On measuring the client-side DNS infrastructure, In Proceedings of the 2013 conference on Internet measurement conference, ACM (2013).

[141] S. Schrauger, *The story of how WoSign gave me an SSL certificate for GitHub.com*, https://www.schrauger.com/the-story-of-how-wosign-gave-me-an-ssl-certificate-for-github-com (2016).

[142] Sharon Goldberg, *The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp*, https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278 (2018).

[143] Sharon Goldberg, *The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp*, https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278 (2018).

[144] H. Shulman and M. Waidner, Fragmentation Considered Leaking: Port Inference for DNS Poisoning, In Applied Cryptography and Network Security (ACNS), Lausanne, Switzerland, Springer (2014).

[145] H. Shulman and M. Waidner, Towards Security of Internet Naming Infrastructure, In European Symposium on Research in Computer Security, Springer (2015).

[146] H. Shulman and M. Waidner, One Key to Sign Them All Considered Vulnerable: Evaluation of DNSSEC in the Internet., In NSDI (2017).

[147] SolarWinds Worldwide, LLC, *GNS3 - The software that empowers network professionals*, https://www.gns3.com (unpublished).

[148] S. Son and V. Shmatikov, The Hitchhikers Guide to DNS Cache Poisoning, *Security and Privacy in Communication Networks* 466–483 (2010).

[149] Stanford University, *Stanford Internet Research Data Repository*, https://scans.io/ (unpublished).

[150] E. Syta, P. Jovanovic, E.K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M.J. Fischer, and B. Ford, Scalable bias-resistant distributed randomness, In Security and Privacy (SP), 2017 IEEE Symposium on, Ieee (2017).

[151] P. Szalachowski, S. Matsumoto, and A. Perrig, PoliCert: Secure and flexible TLS certificate management, In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM (2014).

[152] Team CYMRU, *IP to ASN Mapping Service*, https://team-cymru.com/community-services/ip-asn-mapping/ (unpublished).

[153] The TOR project, *Consensus Health*, https://consensus-health.torproject.org (unpublished).

[154] The TOR project, *Tor Protocol Specification*, https://2019.www.torproject.org/docs/faq.html.en#ChoosePathLength (unpublished).

[155] The TOR project, *Tor Protocol Specification*, https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt (unpublished).

[156] X. Wang, A.J. Malozemoff, and J. Katz, *EMP-toolkit: Efficient MultiParty computation toolkit*, https://github.com/emp-toolkit (2016).

[157] X. Wang, S. Ranellucci, and J. Katz, Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation, In CCS (ACM, 2017).

[158] D.X. Wei, C. Jin, S.H. Low, and S. Hegde, FAST TCP: motivation, architecture, algorithms, performance, *IEEE/ACM Trans. Netw. 16*(6), 1246–1259 (2006).

[159] D. Wendlandt, D.G. Andersen, and A. Perrig, Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing., In USENIX Annual Technical Conference (2008).

[160] H. Wu, Z. Feng, C. Guo, and Y. Zhang, ICTCP: Incast Congestion Control for TCP in data center networks, In CoNEXT (ACM, 2010).

[161] H. Wu, Z. Feng, C. Guo, and Y. Zhang, ICTCP: Incast congestion control for TCP in data-center networks, *IEEE/ACM Transactions on Networking (ToN) 21*(2), 345–358 (2013).

[162] W3Techs, *Usage statistics of SSL certificate authorities for websites*, https://w3techs.com/technologies/overview/ssl_certificate (2018).

[163] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, Deploying cryptography in Internet-scale systems: A case study on DNSSEC, *Dependable and Secure Computing, IEEE Transactions on 8*(5), 656–669 (2011).

[164] A.C.C. Yao, How to Generate and Exchange Secrets (Extended Abstract), In FOCS (IEEE Computer Society, 1986).

[165] E.A. Yfantis and J.B. Pedersen, Random number generators: Metrics and tests for uniformity and randomness, *JCMCC. The Journal of Combinatorial Mathematics and Combinatorial Computing 74* (2010).

[166] S. Zahur, M. Rosulek, and D. Evans, Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates, In EUROCRYPT (2) (Springer, 2015).