TECHNISCHE
UNIVERSITÄT
WIEN
Vienna University of Technology

Dissertation

Trading Dependability, Performance, and Security in First-Price Sealed-Bid Online Auctions with Temporal Decoupling

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.Prof. Dr. Schahram Dustdar und
Dr.techn. Karl M. Göschka
E184-1
Institut für Informationssysteme
Distributed Systems Group

und

Prof. Dr. Engin Kirda
College of Computer and Information Science
Northeastern University

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Dipl.-Ing. Mag. Günther Starnberger
Matr. Nr. 0126066
Allerheiligengasse 5/2/1, 1230 Wien

Wien, Februar 2011 _____

Kurzfassung

Bei Erstpreisauktionen mit verdeckten Geboten, wie sie zum Beispiel für Staatsanleihen durchgeführt werden, treten üblicherweise hohe Lastspitzen kurz vor dem Auktionsende auf, da ein spätes Bieten es den Bietern erlaubt ihr Gebot besser an die jeweilige Marktsituation anzupassen. Weiters gibt es hohe Anforderungen an die Zuverlässigkeit, da Ausfälle des Auktionsservers zu signifikanten finanziellen Verlusten führen können, falls sich bis zur Wiederholung der Auktion die Situation an den Finanzmärkten erheblich verändert. Aus diesem Grund müssen Systeme für hohe Lastspitzen und hohe Verfügbarkeit ausgelegt werden, was zu entsprechend hohen Kosten führt. Gerade in Bereichen, in denen Auktionen nur wenige Male pro Jahr durchgeführt werden, wie zum Beispiel Auktionen von Staatsanleihen, ist so eine Lösung nicht praktikabel, da die Systeme während der restlichen Zeit nicht verwendet werden. In vielen Fällen sind Alternativen wie Cloud-Computing aufgrund der gegebenen Sicherheitsanforderungen ebenfalls keine Lösung.

Grundidee hinter dem Lösungsansatz dieser Dissertation ist es, hohe Verfügbarkeit zu erreichen, indem ein Teil der vorhandenen Sicherheit gegen Verfügbarkeit eintauscht wird. In einem zweiten Schritt werden dann die neu entstandenen Sicherheitsprobleme gelöst. Zuerst wird die Verfügbarkeit erhöht, indem einzelne Komponenten des Systems zeitlich voneinander entkoppelt werden. Dadurch können Bieter Gebote direkt an eine sichere Smartcard übermitteln, und erst zu einem späteren Zeitpunkt zum Server übertragen. Allerdings entstehen dadurch neue Sicherheitsprobleme, da Angreifer neue Möglichkeiten erhalten das System zu attackieren. Diese Probleme werden durch Beiträge in folgenden Bereichen gelöst: (i) Adaptiver Regelung, (ii) sichere Zeitsynchronisation und Zeitstempel und (iii) sichere Kommunikation und Transaktionsauthentifizierung.

Das Ergebnis unserer Evaluierungen zeigt, dass unsere zeitliche Entkopplung es erlaubt, den Einfluss von Lastspitzen auf die Server entscheidend zu verringern und damit die Anzahl der pro Server unterstützbaren Bieter zu erhöhen. Weiters ist es möglich, die Gebote nicht nur in der zeitlichen Ebene gleichmäßig zu verteilen, sondern auch die Gesamtanzahl der Geboten zu reduzieren, da in bestimmten Fällen später abgegebene Gebote frühere Gebote überschreiben können. Unser Interval-basierendes Zeitstempelprotokoll verhindert, dass Zeitstempel von Bietern manipuliert werden können, wodurch diese einen Vorteil gegenüber anderen Bietern erhalten würden. Weiters gewährleistet unser Authentifizierungsverfahren die Sicherheit während einer Auktion in Fällen, in denen ein Bieter einen unsicheren Computer verwendet, welcher möglicherweise mit Malware infiziert ist. Dies ermöglicht Nachweisbarkeit von Geboten, da es aufgrund der Sicherheitsmaßnahmen für Bieter schwieriger wird abzustreiten ein bestimmtes Gebot platziert zu haben.

Abstract

First-price sealed-bid auction scenarios generally exhibit high peak loads around the auction deadline as a majority of bidders tries to submit bids shortly before the deadline. Moreover, these auctions also exhibit high dependability requirements as an auction canceled due to service failures can lead to significant financial losses, because of financial market conditions changing over time. As a consequence, systems need to be designed for excessive workloads and high availability, leading to massive over-provisioning and high costs. However, for some governmental auctions, such as bond auctions and CO_2 certificate auctions, this over-provisioning is not cost efficient as auctions are only conducted a few times a year with the hardware being idle during the remaining time. Moreover, cloud computing solutions are not an option due to security issues such as data ownership.

Key idea of our approach is to alleviate the dependability problems by shifting them into the security domain and by consequently solving the new security problems. We increase dependability of the system by temporally decoupling the individual components of the system from each other and allowing users to place bids on trusted devices physically located at their place. However, by doing so we decrease the security as we give adversaries new options to attack the system. To address those issues, we provide contributions in the areas of (i) adaptive rate control, (ii) secure time synchronization and timestamping, and (iii) secure communication and transaction authentication, as those areas are primarily affected by our temporal decoupling approach.

Evaluation of our contributions shows that our temporal decoupling approach is able to considerably reduce the impact of peak load on servers and thereby increase the system's performance. We are not only able to distribute requests in the temporal domain, but due to the fact that some requests can deprecate earlier requests by the same client we can also reduce the overall amount of transmitted information. In addition, our secure time synchronization and timestamping protocol prevents bidders from manipulating timestamps and thereby ensures fairness of auctions. Finally, our secure communication and transaction authentication approaches provide security of transactions in cases where untrusted clients are used. Thereby, those contributions also provide non-repudiation, as they prevent malware on the local terminal from successfully manipulating bids.

Acknowledgments

First and foremost, I would like to thank my advisors Schahram Dustdar, Karl M. Göschka, and Engin Kirda for the great opportunity to carry out my thesis at the TRADE project, their excellent guidance of my dissertation during all phases of my research, and the experience gained in the research project.

I greatly appreciate the feedback of Karl M. Göschka and Lorenz Froihofer. Their valuable comments and suggestions helped me to improve this thesis. Further thanks go to Engin Kirda and Christopher Krügel for establishing the link to Karl M. Göschka and the TRADE project.

I am especially thankful to the master's students who contributed to this thesis with prototype implementations and valuable feedback concerning my research contributions: Markus Jung, Juraj Holtak, and Markus Wilthaner.

Thanks go to Thomas Obereder and Gerald Müllan of IRIAN for providing input and feedback regarding use of the contributions in commercial applications scenarios.

Finally, I would like to thank the Austrian Federal Ministry of Transport, Innovation and Technology for funding part of this work under the FIT-IT project TRADE (Trustworthy Adaptive Quality Balancing through Temporal Decoupling, contract 816143, `http://www.dedisys.org/trade/`).

Contents

Glossary

AES Advanced Encryption Standard. 73, 92

AID Application Identifier. 86

AIK Attestation Identity Key. 90, 91, 93

APDU Application Protocol Data Unit. 6, 46, 47, 49, 86–88, 92, 93, 95–97, 105

API Application Programming Interface. 47–49

ATR Answer To Reset. 86, 87, 97

CA Certificate Authority. 86, 91, 93

CPU Central Processing Unit. 24, 25

DH Diffie-Hellman. 90

DHT Distributed Hash Table. 56, 57, 59

DOS Denial Of Service. 94

ECC Elliptic Curve Cryptography. 74

EK Endorsement Key. 91

FLL Frequency-Locked-Loop. 50

FPGA Field Programmable Gate Array. 69

GB Gigabyte. 69

GSM Global System for Mobile Communications. 69

HMAC Hash-based Message Authentication Code. 91, 92

HTTP Hypertext Transfer Protocol. 24, 25, 29, 30, 97, 98

I/O Input/Output. 89

ID Identification Data. 12, 30, 56, 57, 60, 61, 118

IEC International Electrotechnical Commission. 92

IETF Internet Engineering Task Force. 97

IP Internet Protocol. 57

ISO International Organization for Standardization. 92

IV Initialization Vector. 93

Java SE Java Platform, Standard Edition. 49

JCOP Java Card OpenPlatform. 47, 49

JSR-177 Security and Trust Services API for J2ME. 105

MD5 Message-Digest algorithm 5. 47

mTAN Mobile Transaction Authentication Number. 73, 79, 80

MV Manipulated Variable. 21

NAT Network Address Translation. 54, 55, 62

NTP Network Time Protocol. 38, 41, 46, 47, 50, 51, 104

NXP NXP Semiconductors. 47

OMA Open Mobile Alliance. 97

P Proportional. 20, 21

PC Personal Computer. 25, 46, 76, 81, 82, 87, 91, 93

PCR Platform Configuration Register. 91, 93

PDF Portable Document Format. 76

PI Proportional-Integral. 20, 34, 35

PID Proportional-Integral-Derivative. 4, 8, 10, 11, 19–21, 25, 27, 36, 102

PIN Personal Identification Number. 78

PLL Phase-Locked-Loop. 38, 50

PV Process Variable. 20

QoS Quality of Service. 34

QR Quick Response. 67, 70, 72, 74, 77, 78, 80, 96, 97, 104

QR-TAN Quick Response - Transaction Authentication Number. 5, 8, 9, 66, 67, 70–73, 76–83, 85, 94–98, 102–105

R-UIM Removable User Identification Module. 98

RADclock Robust Absolute and Difference Clock. 50

RPC Remote Procedure Call. 86, 87

RTC Remote Trusted Computer. 94

RUBiS Rice University Bidding System. 24

SCWS Smartcard Webserver. 97, 98

SHA-1 Secure Hash Algorithm 1. 73

SIM Subscriber Identification Module. 78

SMS Short Message Service. 69, 78, 79

SP Setpoint. 21

SRDI Shared Resource Distributed Index. 59

SRTT Smoothed Round Trip Time. 30

TAN Transaction Authentication Number. 8, 67–69, 73, 77, 78

TCP Transmission Control Protocol. 6, 29, 30, 34, 35, 64

TEM Trusted Execution Module. 105

TLS Transport Layer Security. 88, 97

TPC-W TPC Benchmark W. 24

TPM Trusted Platform Module. 46, 81–83, 85, 88–91, 93, 98, 103

TXT Trusted Execution Technology. 89

UDP User Datagram Protocol. 6, 62, 78

UICC Universal Integrated Circuit Card. 98

URL Uniform Resource Locator. 31, 87

USB Universal Serial Bus. 78

USIM Universal Subscriber Identification Module. 98

WAN Wide Area Network. 25

Chapter 1

Introduction

First-price sealed-bid auctions [MM87] as used for state bonds generally exhibit high dependability requirements and high peak loads and therefore require systems designed for excessive workloads and high availability. This thesis contributes with techniques to trade dependability, security, and performance, allowing to provide such characteristics without changes to the server infrastructure. Key idea of the approach followed in this thesis is to alleviate dependability problems by shifting them into the security domain and by consequently solving the new security problems [FG08]. We increase dependability of the system by temporally decoupling the individual components of the system from each other, and allowing users to place bids on trusted devices physically connected to their computer. However, by doing so we decrease the security of the system, as we give adversaries new options to attack the system. To address these issues we designed and implemented a secure timestamping protocol that allows us to assign accurate timestamps to bids when they are placed and—in the case of high peak loads or temporary outages—to queue the bids at the client and transfer them to the server at a later point in time.

One security problem with client-side timestamping is that software running on clients cannot be protected against attacks by malicious users. Manipulating the timestamps would enable a user to place bids even after the auction's deadline, and to gain potential advantages from information available after the deadline that may influence the user's decision on the bid. By shifting the timestamping process from an untrusted computer to a secure device we can prevent users from tampering with critical parts of the software. However, only securing the software is not sufficient as users still retain full control over the network between the secure device and the auctioneer. As a

consequence, malicious users can selectively delay packets and cause deliberate offsets during time synchronization. Our timestamping protocol mitigates these issues by enhancing on existing interval-based time synchronization techniques.

In addition to the protection of auctions against potentially malicious bidders, we also need to ensure the security of honest bidders. Therefore, we introduce two security extensions facilitating secure transaction authentication and secure smart card communication.

Summarized, the main contributions of this thesis are in the areas of (i) adaptive rate control and performance, (ii) time synchronization and timestamping, and (iii) secure communication and transaction authentication. The contributions in the first area allow us to increase the dependability of the system at the cost of security, while the contributions in the subsequent two areas solve the resulting security problems.

In this chapter we first introduce our application scenario, followed by a discussion of the structure and the contributions of this thesis.

1.1 Application scenario

The introduction of our application scenario is partitioned into four main parts: First, we discuss the characteristics of first-price sealed-bid auctions, followed by an examination of our temporal decoupling approach. We then proceed with a discussion about our system architecture and conclude the section with an introduction to our trust model.

1.1.1 First-price sealed-bid auctions

In first-price sealed-bid auctions the time of bid placement does not have an influence on the auction outcome, as long as bids are placed before the fixed auction deadline. Moreover, bidders do not learn about each others' bids until the end of the auction and are allowed to submit updates to their bids until the auction deadline. Examples of such first-price sealed bid auctions are governmental bond auctions and CO_2 certificate auctions.

In this thesis we consider first-price sealed-bid auctions as used for the auctioning of bonds. These auctions exhibit high peak loads around the auction deadline as a majority of bidders tries to submit bids shortly before the deadline. Moreover, these auctions also exhibit high dependability requirements as

an auction canceled due to technical reasons can lead to significant financial losses, because financial market conditions change over time. Unlike in the case of eBay-style auctions, only few non-overlapping auctions are conducted per year. Therefore, it is not possible to schedule multiple auctions in a way that the combination of the individual auction peaks produces a more or less constant load, which would lead to good overall server utilization.

As a consequence, systems need to be designed for excessive peak loads and high availability, leading to massive over-provisioning and excessive costs. While cloud services would allow to mitigate some of the availability and performance issues, the high trust requirements, the high monetary amounts, and policy issues eliminate cloud services as an option for deployment.

1.1.2 Temporal decoupling

To address the peak loads and availability issues, we use *temporal decoupling*. The key idea of temporal decoupling is to decouple bid submission at the client from bid transmission to the server. An example is given in Figure 1.1. The grey line depicts the amount of bids placed by the users in a first-price sealed-bid auction, while the black line indicates the amount of bids transmitted to the server. The deadlines for both lines are different. Thus, once the user places a bid locally, it can be transferred to the server at a later point in time, as long as it arrives before the deadline for messages. Due to the longer duration for bid transmission, the peak load can be decreased, as the bids can be spread in the temporal domain.

However, delaying a bid is not possible without a secure means to determine the original time when the bid was placed. We facilitate secure timestamps with a secure time synchronization and a secure timestamping protocol executed on smart cards that are distributed to the users. As users do not have direct access to the software running on the smart cards, it is not easily possible to assign manipulated timestamps to individual bids. After receiving a bid, the server can read the secure timestamp, to ensure that the bid has been placed before the deadline for bids.

By decoupling the deadline for bids from the deadline for message reception at the server (as depicted in Figure 1.1), we increase system performance, availability and scalability during peak loads:

1. We increase performance and scalability, as we can intentionally delay transmission of bids during brief periods of peak loads. This decreases

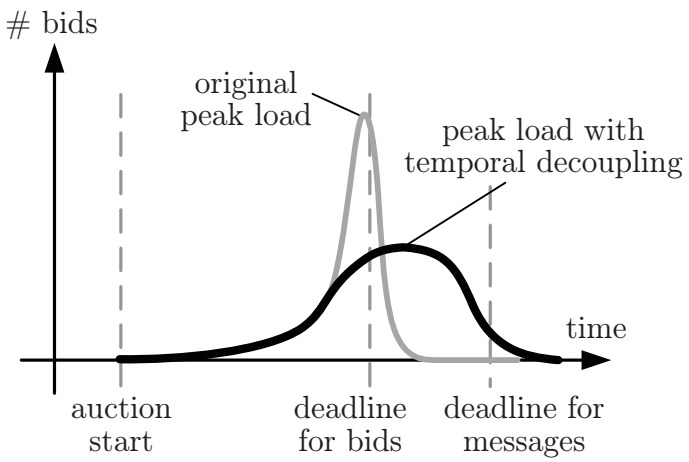


Figure 1.1: Effect of temporal decoupling. The deadline for bids specifies the deadline until which a user can place a bid, while the deadline for messages specifies the deadline until which the bid has to be received at the server.

peak loads at the server and increases the amount of bids that can be placed at clients.

2. We increase availability, as in case of transient problems we can allow bids to be transferred to the server at a later time.

For the availability, it is sufficient to apply correct timestamps locally. If the client is able to communicate with the server, it can immediately transfer bids to the server. Otherwise, it defers transmission until the server becomes available. For performance and scalability we additionally use a decoupling strategy in combination with a PID (Proportional-Integral-Derivative) controller and a distributed feedback channel, which enables an optimal utilization of the server.

While our approach increases the effective performance of the system under peak loads, it also introduces new security challenges, as malicious bidders could tamper with the timestamp, thereby cheating the system. Therefore, we do not define a single global deadline for messages—as in the simplified example in Figure 1.1—but instead define an individual deadline for each single message by restricting the maximum transmission delay of each message. Thus, an attacker trying to manipulate timestamps has less time to tamper with bid information stored on the smart card and cannot assign arbitrary values to timestamps. In case of server failures that prevent clients from abiding to these deadlines, the individual deadlines can be adaptively prolonged on the server-side by the auctioneer.

1.1.3 System architecture

Due to our temporal decoupling approach and due to the smart cards distributed to users, our system architecture is different from traditional auction scenarios. Figure 1.2 depicts the individual components of our system architecture. The smart card is connected to the user's computer that relays messages between smart card and auctioneer. A circumventive user is able to attack different parts of the system. For example, physical attacks can be used against the smart card itself, delay attacks can be used against time synchronization messages relayed by the computer, and any software running on the computer may be manipulated.

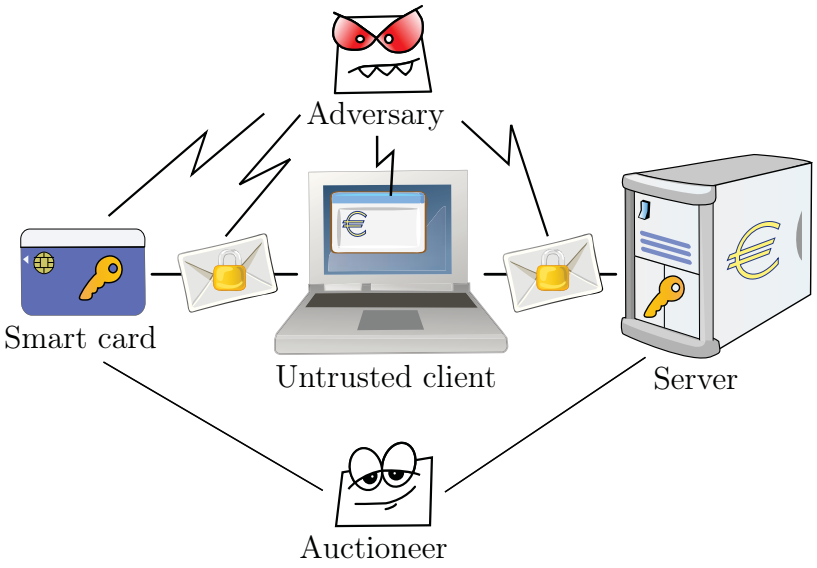


Figure 1.2: Application scenario

- The *auction server* is responsible for hosting the auction Web application and additionally acts as a time server. Bids timestamped by a smart card are relayed over the untrusted client to the auction server.
- The *smart card* timestamps bids provided by the untrusted client with the current time. It obtains the time using a time synchronization protocol. Messages between smart cards and time servers are relayed by the untrusted client in between. Messages between smart cards and bidders are secured with QR-TANs (Quick Response - Transaction Authentication Numbers).

- The *untrusted client* enables communication between smart cards and the auctioneer. Messages transmitted between a smart card and an untrusted client are encoded as APDU (Application Protocol Data Unit), while the untrusted client uses standard Internet protocols, such as the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol), to communicate with the auction server. Only non-security-critical software operations are executed on the untrusted client.

To provide correct time on trusted devices, such as smart cards, we employ a secure time synchronization protocol. The reference time is obtained from servers under control of the auctioneer.

1.1.4 Trust model

In addition to the system architecture discussed in Section 1.1.3 our trust model also differs from traditional time synchronization and timestamping scenarios. While in traditional time synchronization protocols individuals interested in obtaining accurate timestamps have full control over clients, in our scenario the auctioneer has full control over the time servers, but no physical control over the clients. In addition, the user can be considered as a potential adversary with full control over the network.

Figure 1.3 shows the trust relationships in our application scenario. Green solid arrows indicate that the respective role is able to reasonably trust another role or component, if appropriate security mechanisms, e.g., encryption of network links or intrusion detection mechanisms, are applied or contracts between different roles are arranged. Otherwise, the untrusted relationship is indicated via a red dashed arrow.

Based upon existing technology, the auctioneer is able to trust the own infrastructure (auction server, network links, and smart card), the issuer, and the time server infrastructure. Although, we assume the auctioneer may trust the bidder in general, they cannot fully trust the bidder as the manipulation of the clock on the bidder's terminal is undetectable. Consequently, the auctioneer has to distrust the bidder with this respect. Similarly, the auctioneer has to distrust the bidder's terminal and the communication channel between terminal and smart card.

The communication between the smart card and the time server can be trusted with respect to message integrity, but as communication is only performed via the bidder's terminal, communication delays might be introduced by the

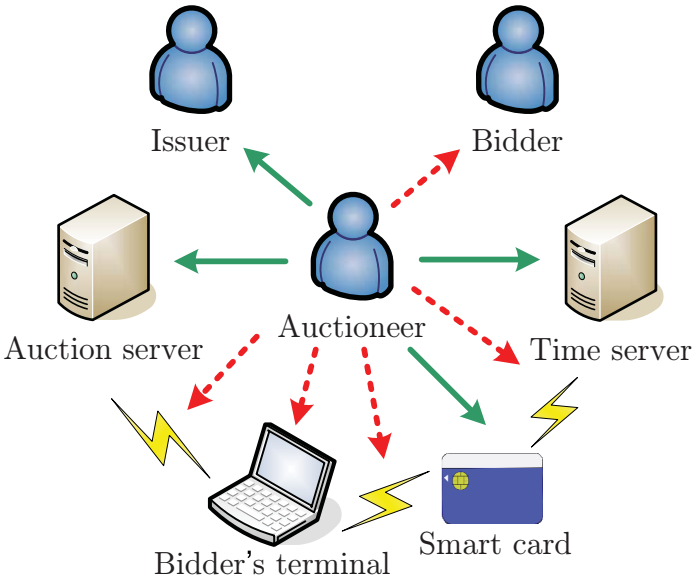


Figure 1.3: Trust model

bidder to influence the time synchronization algorithm. Therefore, this establishes a source of distrust for smart card–timeserver communication. Similar considerations apply to the communication link between the bidder's terminal and auction server as the bidder might interrupt the network connection to pretend a network failure.

1.2 Structure and contributions

This thesis contributes in the areas of (i) adaptive load control and performance, (ii) time synchronization and timestamping, as well as (iii) secure communication and transaction authentication. In particular, the contributions are provided in the following chapters:

Adaptive load control and performance

- Chapter 2 contributes with scalable techniques for client-side request-rate control, combined with actual load reduction for first-price sealed-bid online auctions, especially with a focus on peak load situations [1]. In particular, the main contribution is the integration of (i) a distributed feedback channel to transmit control information from the server to the

clients with (ii) decoupling strategies that allow to constrain client requests directly at the client side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server load. The contributions of this chapter enable our adaptive load control approach and allow us to trade dependability, security, and performance.

Time synchronization and timestamping

- Chapter 3 contributes with a secure time synchronization and timestamping protocol tailored to online auctions where we apply secure timestamps on smart cards locally connected to the bidder's computer [2]. Moreover, our timestamping protocol is robust with respect to man-in-the-middle delay attacks. Finally, we prove the feasibility of our approach based on a .NET smart card implementation and conclude with a discussion of current smart card limitations.

- Chapter 4 contributes with: (i) A distributed timestamp protocol addressing practical applicability issues with an efficient overlay routing architecture able to minimize the effects of node churn and connection establishment delays at the cost of higher impacts of hop-to-hop latencies, (ii) smart card integration to introduce a distributed web of trust and hence increase the security of applied timestamps, and (iii) an evaluation using a prototype implementation and network simulation that shows the performance gains of our protocol in comparison to the state-of-the-art [3]. The properties of our protocol allow for its application in scenarios where distributed timestamping protocols have not been an option so far, for example, because of mutually distrusting users.

Secure communication and transaction authentication

- Chapter 5 contributes with the QR-TAN authentication technique, which is based on two-dimensional barcodes [4]. Compared to other established techniques, such as TANs (Transaction Authentication Numbers), QR-TANs show three advantages: First, QR-TANs allow the user to directly validate the content of a transaction with a trusted device. Second, validation is secure even if an attacker manages to gain full control over a user's computer. Finally, QR-TANs can be used in combination with smart cards instead of a server, facilitating authentication when no network is available. In our application scenario

QR-TANs can increase the security of auctions by preventing malware from successfully manipulating a user's bids.

- Chapter 6 contributes with techniques to Web-enable smart cards and to address the risks of malicious attacks [5]. In particular, our contributions are: (i) A single generic proxy to allow a multitude of authorized Web applications to communicate with *existing* smart cards and (ii) two security extensions to mitigate the effects of malware. Overall, this allows us to mitigate the security risks of Web-based smart card transactions in our application scenario and—at the same time—increases the usability for users.

Chapter 7 discusses alternative application scenarios and concludes the thesis with an outlook on future work.

The publications authored and co-authored by the writer of this dissertation provided in the references above and within the dissertation in numeric references are used in parts of this dissertation without always being referenced explicitly.

Chapter 2

Adaptive rate control

In this chapter we contribute in the area of adaptive load control and performance, by presenting an adaptive rate control technique that facilitates our temporal decoupling approach discussed in Section 1.1.2. In particular, our contribution allows to adjust client transmission rates based on a server's effective load. While temporal decoupling would also be possible without adaptive rate control, adaptivity allows clients to better adapt to a server's resources and a server's load, and to thereby reduce the transmission delay of placed bids. In addition, our adaptive rate control approach allows to increase the security of the system, as it limits the time until which a bid has to be transmitted to the server.

To specify *when* a client is allowed to transmit information, we introduce two decoupling strategies that defer requests directly at the client. The first decoupling strategy specifies which clients are allowed to transmit information at a particular point in time, while the second decoupling strategy controls the transmission rate of clients. For both decoupling strategies the input parameters can be adapted to alter the load at the server. To prevent overload and to enable an optimal utilization of the server we combine the decoupling mechanisms with a server-side PID controller. To minimize the additional load at the server caused due to operation of the controller's feedback channel we use a *distributed* feedback channel operated mainly by the clients themselves. Hence, the server-side cost for operation of the feedback channel becomes independent of the total number of clients, allowing for better performance during peak loads than state-of-the-art systems.

Summarized, our main contribution in this chapter is the integration of (i) a distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain client

requests directly at the client side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization.

First, Section 2.1 presents our decoupling strategies, followed by Section 2.2 which continues with our request rate control loop and the distributed feedback channel. Section 2.3 evaluates the effectiveness of our approach, complemented by Section 2.4 discussing the limitations. Section 2.5 proceeds by reviewing our security vs. dependability trade-off. Afterwards, Section 2.6 compares our adaptive rate control approach to related work. We conclude the chapter in Section 2.7.

2.1 Decoupling strategies

This section provides our first contribution: decoupling strategies and simulations of their expected influence on the clients' transmission rate. A decoupling strategy specifies when and how long a placed bid is held back on the local client. In this section we consider *open-loop control* [KBE99], where fixed input parameters are used for the decoupling strategies. In later sections we extend this open-loop approach by providing *closed-loop* control, which augments open-loop control with (i) a distributed feedback channel (control channel) and (ii) a PID controller to dynamically adapt the input parameters to attain an optimal server utilization.

Each of the examined rate control strategies can be used in two different variants: *Data-overwrite* and *data-queue*. In the *data-queue* variant, all submitted data are queued and eventually transmitted to the server, while in the *data-overwrite* variant, items not yet sent are overwritten by subsequent items—only one item per-auction is queued at a time. The *data-overwrite* strategy is applicable to all temporally decoupled systems where later information obsoletes former information, such as bids in a first-price sealed-bid auction system.

2.1.1 Group-based control

With *group-based* rate control we partition the set of clients into disjoint subsets, where at a given point in time only the clients within a particular subset are allowed to transmit data to the server. There are two distinct steps: (i) Partitioning the original set into a set of disjoint groups and (ii) the selection of an active group. In our approach these two concepts are

intertwined: Each client has a unique ID (Identification Data). When the server wants to select a particular group it broadcasts two values: A *divisor* and a *modulo* value. The divisor specifies how many groups exist, while the modulo value specifies which of these groups is currently active. Each client divides its ID by the divisor and verifies if the congruence class modulo the divisor matches the *modulo* value. If this is the case, the client is within the *active* group. Otherwise, the client needs to wait until its group gets activated.

The modulo value is incremented in predefined time intervals, enabling iteration over existing groups. This interval between groups in combination with the amount of groups determines the maximum possible delay of client requests. In the worst case a client wants to issue a request right at the moment where it became *inactive*. Thus, the client has to wait $interval\_between\_groups \cdot (amount\_of\_groups - 1)$ time units until it is able to transmit the request.

2.1.2 Interval-based rate control

Interval-based rate control exploits the fact that during a peak not only the overall system load increases, but also the transmission rate of individual clients. The idea is to partition time into disjoint intervals with length i and to allow each client to send at most one data item during such an interval. If a client has not transmitted any data to the server in a particular interval, it can transmit the next data item directly to the server without any delay. Otherwise, the client needs to wait for the next interval to be able to send further data. It is crucial that the start points of the intervals differ between clients. Otherwise, all clients with queued packets that wait for the next interval would start submitting their queued packets at exactly the same time.

An advantage of the interval-based approach is that it does not delay data transmission under low load, i.e, when new data items are created at a client in time intervals larger than the submission interval. Only in cases where more than one data item per interval is created, transmission of data will be delayed. This approach can be seen as a special case of token bucket based rate control with a bucket size of one and a new token added each i seconds. To control the request rate of clients, the size of the interval i can be adapted.

2.1.3 Simulation

In this section we use discrete event simulation [HOP+86] to validate the performance benefit of our temporal decoupling approaches. Compared to traditional simulation approaches, discrete event simulation allows for efficient simulation of long time spans within short amounts of time. The simulation uses a priority queue of events sorted after the events' time from which events are iteratively dequeued and processed. When an event is executed, the simulated clock is updated to the value of the event's time. The execution of an event can lead to additional events that are pushed on the priority queue.

First, we generate a load curve that represents when bids are submitted by users during the peak load. Afterwards, we examine the effectiveness of our two temporal decoupling strategies. For each of the strategies we provide performance evaluations for different parameters and both variants (*data-overwriting* and *data-queuing*). Neither the shape of the curves nor the relative load at a particular point in time depend on the total number of clients. Therefore, we give the load relative to the peak load that occurs when no decoupling strategy is used.

Bid behavior

For the simulation of the bidder's behavior we focus on the area around the peak load. Using statistical data obtained from real world auctions of our industrial project partner we observe that $\frac{2}{3}$ of the bids are placed during the peak load in the final five minutes of an auction. We exploit the fact that in a Gaussian distribution 68.27% of all measurements are located within a distance of σ from μ. As these 68.27% roughly correspond to the $\frac{2}{3}$ of the bids, we can assume a value of 150 seconds for σ, to let the Gaussian distribution approximate a peak load over a duration of five minutes. For μ we use a value of zero, placing the peak of the load at the null point of the diagram.

The resulting load curve representing the bidding behavior is shown as grey line in the diagrams in Figures 2.1–2.5. The x-axis represents the time in seconds and the y-axis represents the amount of placed bids per second given in percentage of the peak load of the original load curve, where 100% typically refers to 30 000 bids per second in our simulation. This load curve is used as input for our two different decoupling strategies.

Group-based rate control

For the simulation of the *group-based* strategies we simulated the load with two different sets of input parameters: In the first case we used 20 groups—shown in Figure 2.1—while in the second case we used 75 groups—shown in Figure 2.2. In both cases we switch groups in 4 second intervals. In addition, we simulated the *data-overwriting* behavior (in red with dashed linestyle) and the *data-queuing* behavior (in blue with solid linestyle). The parameters used in the simulation have been chosen according to the constraints—such as the maximum time span between the deadline for bids and the deadline for messages—in real-world first-price sealed-bid auctions.

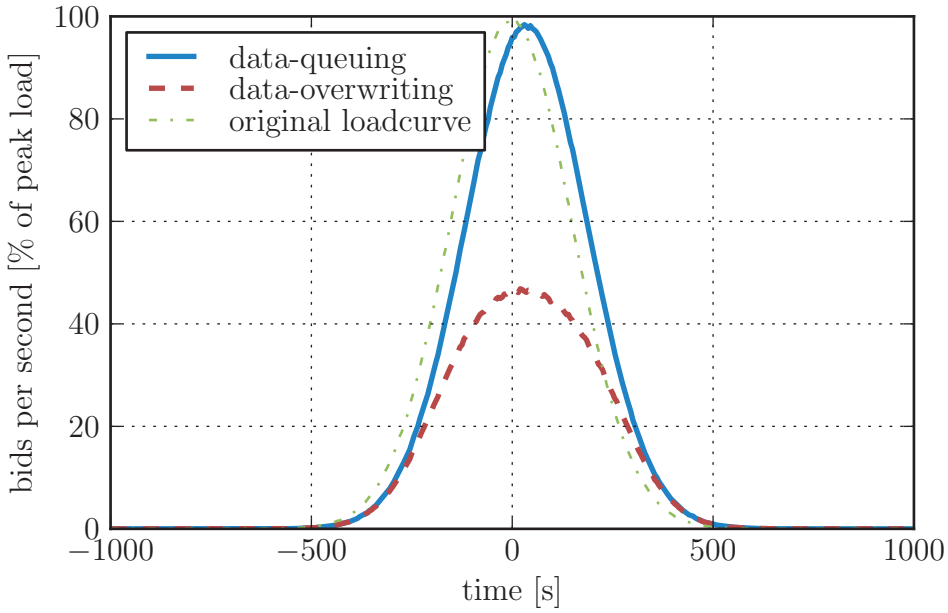


Figure 2.1: Group strategy: 20 groups, iteration each 4 seconds

First, we observe that there are no considerable differences between the original load curve and the decoupled load curves when *data-queuing* is used. The only difference is that in the case of *data-queuing* the load curve is slightly shifted to the right, which is the expected behavior, as the total amount of transmitted bids does not differ and as bid submission is never delayed for more than one full iteration of all groups. However, when *data-overwriting* is used, significant load reductions can be observed, with a peak load less than 50% of the original peak. Again, this is the expected behavior, as at most one bid is queued while the node itself is not active, even if multiple bids are

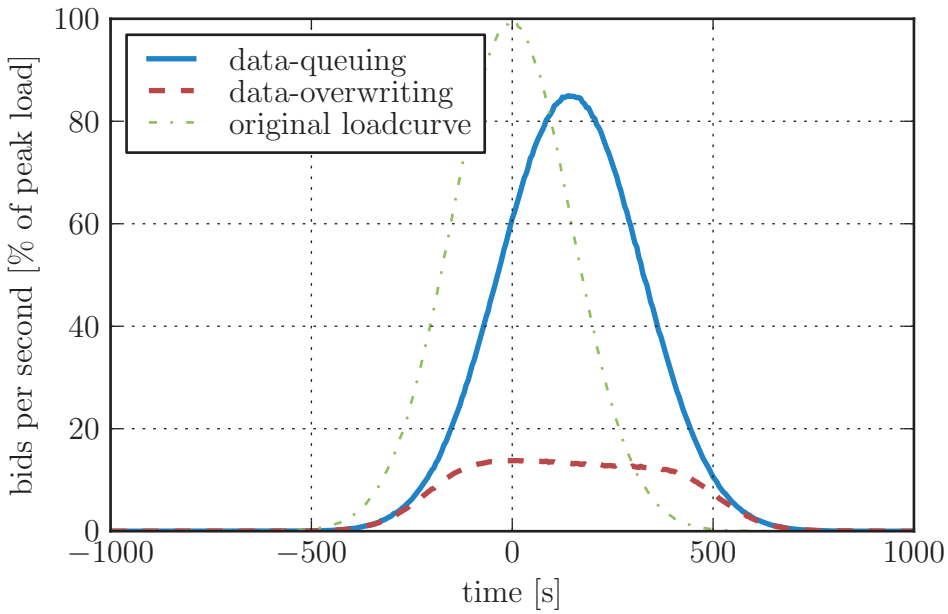


Figure 2.2: Group strategy: 75 groups, iteration each 4 seconds

placed. Rate reduction with *data-overwriting* is only effective, if—on average—more than one bid is placed on a client while it is inactive, as otherwise the resulting rate would be the same as in the data-queuing scenario.

Interval-based rate control

Similarly to the previous strategy, we simulated the results for two different parameters. In the first case we used an interval size of 80 seconds (shown in Figure 2.3), allowing only one bid to be transmitted within an interval of 80 seconds. Bids that cannot be placed immediately are delayed and placed after the interval's end. In the second case we used an interval size of 300 seconds (shown in Figure 2.4). In both cases we simulated the *data-overwriting* and the *data-queuing* variants. As in the case of group-based rate control the parameters have been chosen according to the constraints of real-world first-price sealed-bid auctions. Therefore, the maximum possible delay in the two interval-based data-overwriting cases corresponds to the maximum possible delay in the two group-based data-overwriting cases.

In the results we can observe a brief overshoot and then a relatively stable amount of load. This is the expected behavior: The overshoot occurs as during the increasing overall bid rates some clients have not yet transmitted

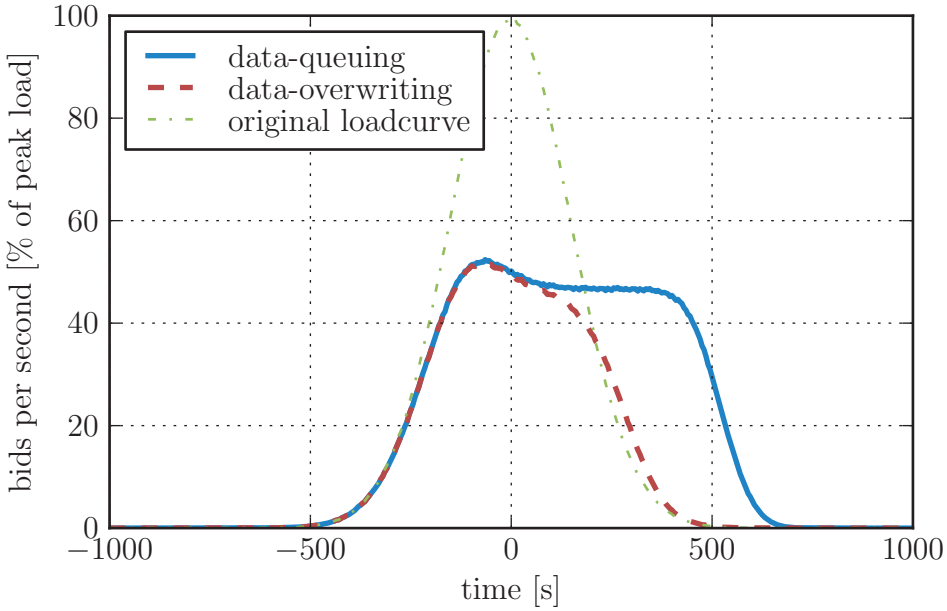


Figure 2.3: Interval limit strategy: 80 seconds

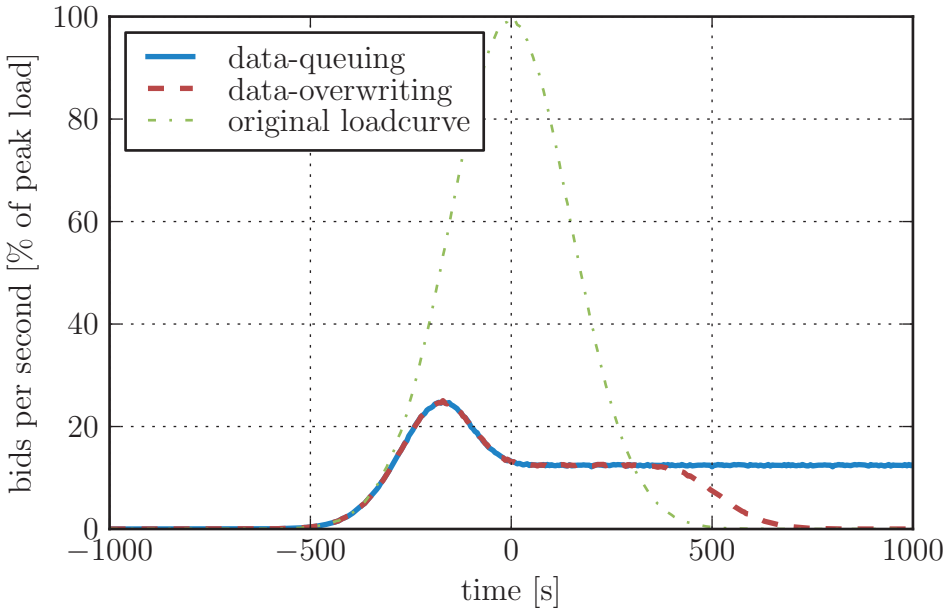


Figure 2.4: Interval limit strategy: 300 seconds

a bid in the current interval. Thus, they can immediately transmit a newly placed bid without any rate restrictions. Once the interval-based rate control sets in, this is not possible as clients typically transmit bids in each individual interval.

In the *data-overwriting* variant, the height and the width of the load curve are comparable to the *group-based strategy* examined in the previous section. However, in the *data-queuing* variant, there is a fundamental difference between the *group-based strategy* and the *interval limitation* strategy: While in the group-based strategy *data-queuing* leads to a higher load when compared to the *data-overwriting* case, in the *interval limitation* strategy, *data-queuing* does not show a higher peak load, but an increased duration. Again, this is the expected behavior, as interval-based rate control restricts the possible transmission rate. Thus, in *data-queuing* the larger amount of bids leads to additional transmission delays. With an interval size of 80 seconds, we can observe that the decoupled curve shows a peak at around 50% of the original curve—which essentially doubles the amount of users a system can handle. With larger delays or intervals the peak load can be even further decreased. However, larger interval sizes also imply larger time spans between the deadline for bids and the earliest possible deadline for messages.

Transmission failures

Figure 2.5 shows the performance of interval-based rate control in case of transmission failures. In the depicted scenario, bids cannot be transmitted to the server between the time values 0 and 180 on the x-axis. For the simulation an interval size of 80 is used. The *re-transmission* strategy of a client is to try again when the next local interval starts. In the *data-overwriting* variant the only difference to Figure 2.3 is that the transmission rate drops to zero during the outage. As each client can have at most one outstanding bid, the point until all bids have been received at the server is not substantially deferred. In the *data-queuing* variant each single bid needs to be transferred to the server. Therefore, the duration until all bids have been collected at the server is prolonged by approximately the duration of the outage. Due to the re-transmission strategy, the transmission rates of clients do not change when the server is not available. In the figure this can be observed by the server's load instantly reaching its *normal* value as soon as transmission is possible again.

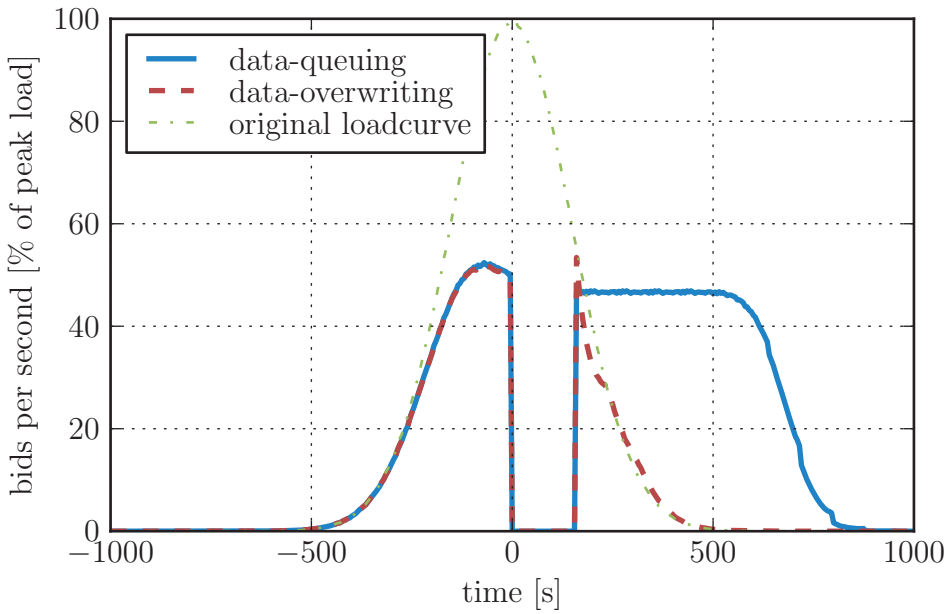


Figure 2.5: Transmission failure mitigation

Results

Our results show that the group-based and interval-based decoupling strategies are suitable for temporal decoupling to mitigate temporary peak loads. The *group-based* strategy allows to mitigate peak loads by assigning nodes to a set of groups and allowing nodes to transmit data only when their particular group is active. We have evaluated this strategy with two different variants: *Data-queuing* and *data-overwriting*. As bids that are placed while the node's respective group is inactive are delayed, both variants also slightly shift the load curve to the right. However, for a given delay parameter the data-overwriting variant is able to reach a considerably lower peak load then the data-queuing variant.

The idea behind the *interval limit* strategy is to limit the maximum transmission rate of each single client. Therefore, from a bidder's point of view bids are not delayed until the local transmission rate exceeds the allowed transmission rate. For both variants—*data-queuing* and *data-overwriting*—the resulting peak load is the same for a given parameter. However, the time until all bids have been collected at the server is longer when *data-queuing* is used, due to the larger amount of total bids.

The simulations in this section used predefined input parameters, such as the amount of groups or the interval size. While those input parameters allow to reduce request rates, they are open-loop control approaches that do not take actual load at the server into account. Thus, these approaches are not able to prevent server overload if input parameters are suboptimally chosen, e.g., if the amount of active clients changes over time. Moreover, open-loop control only achieves a sub-optimal server utilization. In the next section we upgrade this open-loop approach to a closed-loop approach, allowing for an optimal server utilization.

2.2 Closed-loop control

In this section we contribute with our closed-loop control approach that integrates our decoupling strategies with (i) a distributed feedback channel and (ii) a PID controller to induce an optimal utilization at the server. The PID controller dynamically adapts the input parameters of our decoupling strategies, while the distributed feedback channel can relay control information to the clients with only minimal overhead at the server.

The advantage of closed-loop control in comparison to the open-loop strategies presented in Section 2.1 is that the PID controller allows the system to adapt to changing transmission rates of clients, as well as a changing number of overall clients. While open-loop decoupling strategies can adapt to the worst-case considering the highest potential overall server load, this unnecessarily delays bid transmission at the clients at times where the system is not fully loaded. Consequently, malicious bidders would have more time to tamper with the stored bids, thereby increasing the system's security threats.

System and software architectures are depicted and described in Figure 2.6. First, we discuss the setup at the client-side. Then we proceed with a discussion of the controller. Finally, we conclude the section with our distributed feedback channel.

2.2.1 Clients

On the client-side we use a decoupling application to facilitate request rate control. This decoupling application performs two tasks: (i) It acts as actuator by receiving control information over the distributed feedback channel and by controlling request rates of the Internet application accordingly, and (ii) it is itself part of the distributed feedback channel as it forwards control

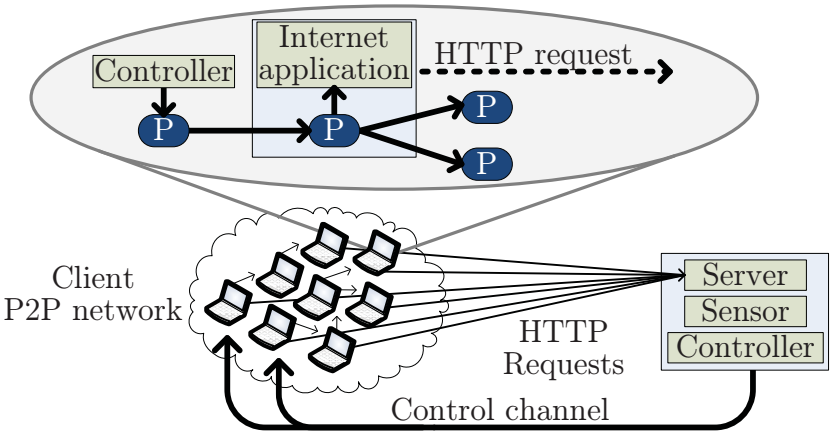


Figure 2.6: Control loop architecture overview. The individual clients send requests to the server. At the server the sensor measures the server-side queuing and processing delay and the controller uses the input provided by the sensor to adjust the output, e.g., the interval length or group size of our decoupling strategies. A distributed feedback channel is used to scalably propagate the control output back to the clients. On each client the Internet application originates the application-level requests, while the decoupling application (labeled P) receives input from the controller over the distributed feedback channel and provides input to other peers and the Internet application.

information to other clients. Due to our closed user group, installation of custom client-side software is a feasible approach. As a consequence, we do not target generic Internet applications, but specific types of applications such as first-price sealed-bid auctions that benefit from temporal decoupling as outlined in the introduction in Section 1.1.2.

2.2.2 Controller

The task of the controller discussed in this section is to adaptively control the parameters of the rate control techniques introduced in Section 2.1. We use a *closed-loop* control approach that allows the controller to provide feedback to individual clients and thereby provides a better performance in case of unpredictable loads and unanticipated load changes.

The setup of our controller is depicted in Figure 2.7. The controller is implemented as PID controller, as measurements showed that it provides a lower standard deviation of the PV (Process Variable) than a P (Proportional) or PI (Proportional-Integral) controller in our application scenario. Clients

regularly send requests to the server. The input to the controller is provided by a sensor at the server that measures the *process variable*. The controller then compares the *error* (or deviation) e between the given SP (Setpoint) and the measured process variable. The goal of the controller is to minimize e by adapting the input parameters of the system accordingly. In our system the process variable is—depending on the configuration—either the queue length of requests waiting to be processed, or—alternatively—the request processing delay at the server. Consequently, the setpoint is the target value of the system and given in the same unit as the process variable. The MV (Manipulated Variable) is the input to the process, in our system this corresponds to the input parameters of the decoupling mechanisms introduced in Section 2.1. In case of group-based decoupling this parameter reflects the percentage of active clients, while in the case of interval-based decoupling the parameter reflects the interval size.

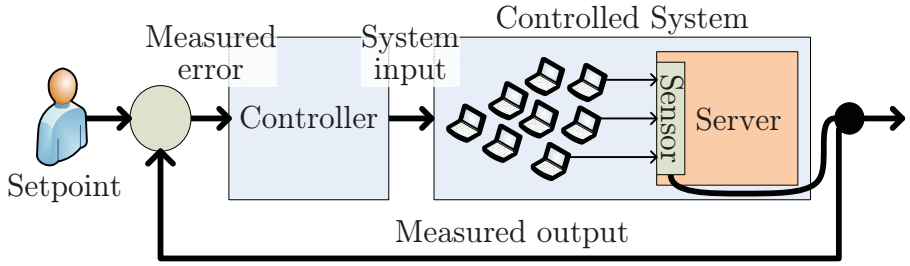


Figure 2.7: Control loop. Our controlled system consists of the individual clients and the server. A sensor at the server measures the processing delay. The measured error between an externally provided setpoint and the measured output is fed into the controller, which adjusts the system input accordingly. The system input is provided via a distributed feedback channel back to the clients.

Due to the fact that the system behavior differs between application scenarios and as we anticipate that in real-world deployments at best a heuristic tuning or auto-tuning method will be used, we use the Ziegler-Nichols heuristic method [ZN42, HÅ02] to obtain P, I, and D gains with a quality that can be compared to such real-world deployments. First, the P gain is increased until the system starts to oscillate. Then, the Ziegler-Nichols charts are used to calculate the respective gains for a PID controller.

2.2.3 Distributed feedback channel

The task of the distributed feedback channel is to transmit control information from the controller to the clients. The typical implementation approach is to let the server communicate with each individual client. However, previous work has shown that for large amounts of clients such a system cannot be easily implemented without a specialized broadcasting infrastructure [HJ99].

Consequently, we contribute with the integration of a broadcasting infrastructure based on an overlay network that can be used to *flood* information to the set of clients. We use application-level broadcast [YLE04] based on a tree-based network structure as depicted in Figure 2.8. Therefore the height of the tree—and thereby the propagation delay—grows only logarithmically with the amount of clients. To prevent failures caused by single clients, we use multiple independent trees, so that each client receives broadcasts from multiple sources. To detect partitions we use a keep-alive mechanism. The server regularly broadcasts cryptographically secured sequence numbers to the tree. A client can detect problems, if it has not received sequence numbers for some time or if individual sequence numbers are missing. However, due to the closed user group in our application scenario we expect a low node churn.

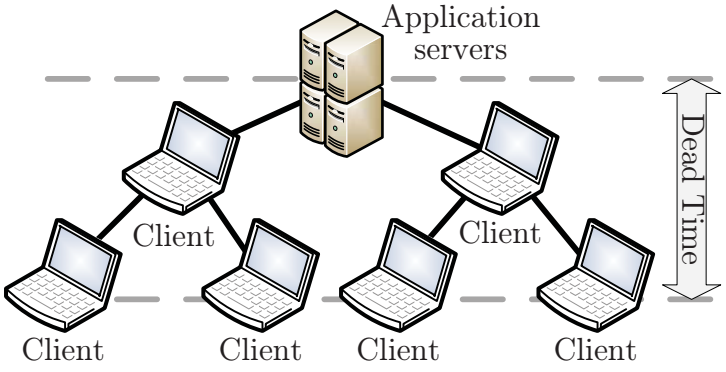


Figure 2.8: Distributed feedback channel. The servers send information to only a few clients with a tree-based network structure used to propagate data between clients. Parameters of the tree are the arity and the height.

For our distributed broadcast channel there is a trade-off between the total number of clients and the stability of our system:

- With a growing number of clients the height and/or arity of the tree have to increase to accommodate more clients. However, with a growing height or arity the dead time of our system increases as well, which in turn leads to instabilities such as oscillation.

- With increasing height of the tree, the clients' average distance to the server increases. Thus, it takes longer until a server broadcast is received by a majority of the clients. Similarly, with increasing arity each client has to transmit information to a larger amount of children, lowering the bandwidth available for each individual transmission.

- When the dead time of the system increases, this decreases the stability, making the system susceptible to overshoots or oscillation. It is possible to mitigate this issue by decreasing the gains of the controller. However, this slows down the system, potentially preventing it from responding to load changes in time.

By adapting the arity and the height of the N-ary tree, we can control the trade-off between the maximum amount of clients and the dead time. For example, consider an average propagation delay between two clients of 100 ms and a maximum tolerable dead time of 5 seconds. This would allow a total tree height of 50. When using a binary tree, this equals a theoretic total of $22.5 \cdot 10^{14}$ clients. In practice, the number of clients is therefore not limited by our broadcast channel, but by the lower bound of the request rate acceptable for each individual client. Under most configurations, the feedback channel itself is capable of supporting an amount of clients several dimensions higher than the maximum amount reasonably supported by the servers of the system. As a consequence, we can use the additional capacity to increase the redundancy of our tree, allowing for correct propagation of broadcasts in cases where individual clients fail.

As a majority of bidders generally places bids shortly before the auction deadline and as the set of bidders is pre-defined for each auction, node churn during this critical period is typically smaller than in comparable application scenarios—if not non-existent at all. Therefore, our distributed feedback channel is not specifically adapted to high node-churn scenarios. However—depending on the concrete scenario—alternative application-level broadcast protocols can be used for such cases.

2.3 Evaluation and measurements

In this section we evaluate the effectiveness of our rate control approach. The goal of the measurements is to evaluate different system configurations in regard to stability and performance. While there are several common benchmarks for evaluating Web applications, they are not applicable to

our temporally decoupled system. RUBiS (Rice University Bidding System) [CMZ02] models an auction site similar to eBay. While in RUBiS the typical user interactions such as browsing of auctions and consulting of bid histories are modeled, in our system the focus lies on the request rate control of temporally decoupled bid submissions. Similarly to RUBiS, TPC-W (TPC Benchmark W) [Tra02] models a Web shop to test the performance of Web server and database systems. As a consequence, neither RUBiS nor TPC-W can be used in their current specification to evaluate our rate control system, as they are not applicable to temporally decoupled systems. Instead, we evaluate our system in the auction scenario specified in Section 2.3.4.

In the following evaluations we first examine the open-loop behavior of our system and the system's response to changes in the input. Then, we examine the closed-loop behavior of our controller in a simulated auction scenario using group-based and interval-based control. In addition, we also compare our approach to alternative solution approaches, such as an approach piggybacking control information in HTTP (Hypertext Transfer Protocol) requests and another approach rejecting requests on the server-side [ASB02].

2.3.1 Configuration

In the evaluation we focus on single-threaded clients using *data-queue* for transmission and the processing and queuing delay as metric. In addition, we also conducted measurements where we limited the number of concurrently processed requests at the server to 1, and subsequently used the size of an unbounded queue of the waiting requests as a metric for the controller. Such a setup better reflects application servers with a fixed number of worker threads. However, as we did not observe significant differences in comparison to the processing and queuing delay metric, we only show the results for the processing and queuing delay metric in this section, which also reflect our results for the queuing size metric.

2.3.2 Test setup

For the evaluation, we measure how our controller works for a CPU (Central Processing Unit) bound service, as this has been indicated by our industrial partner to be the limiting factor. Our service performs a simple calculation—prime factorization—that requires about 100 ms of processing time. An increasing concurrency rate also increases the processing delay for each indi-

vidual request. The hardware setup consists of six standard PCs (Personal Computers) running on Ubuntu 8.04 server edition. The first PC hosts the server, while the remaining PCs act as load generators. The server is implemented as Java application using Jetty to provide HTTP access. The clients are also implemented in Java with one thread per simulated client. Each load generator is responsible for the simulation of multiple clients. Due to the fact that we test a CPU bound service, local network performance is not an issue.

As all clients run on the same local network and as multiple clients run on the same host, the propagation delay between individual clients is low. To account for the fact that dead time has a considerable influence on the performance of a controlled system, we implemented *dead time simulation* into the distributed broadcast channel used to propagate control information to clients. This simulation works by enabling a client to hold feedback information for a particular amount of time before forwarding this information to the next client. This allows to simulate different network conditions, such as latency on WANs (Wide Area Networks).

2.3.3 Response to request rate change

First, we evaluate how fast our system reacts to sudden changes of the request rate. The evaluation was done in two distinct steps depicted in Figure 2.9.

In the "No Control" curve the behavior of the uncontrolled system is shown and in the "PID Controller" curve our system is extended with a PID controller that provides closed-loop control. In both cases we double the amount of active clients and verify how the system reacts to this input change. In the "No Control" case, doubling the amount of clients also doubles the processing time at the server. In the "PID Controller" case, the controller aims to keep the response time stable at 1 000 ms. We can see a peak at the beginning, when the controller needs to find an optimal range at the start of the simulation, and during the change, when the controller needs to adapt group sizes to the new number of clients accordingly (see circles).

Stability of the controller is a considerable factor, as it does not only affect the processing time at the server, but also the total throughput of the system. With a low stability caused by an unstable PID controller, there are time periods where no clients send requests, although the server is not fully loaded. In addition, also the case where too many requests are sent by clients can cause decreased throughput due to overload at the server. Generally, the goal of a controller is to reach the best performance possible with a given limit for the stability.

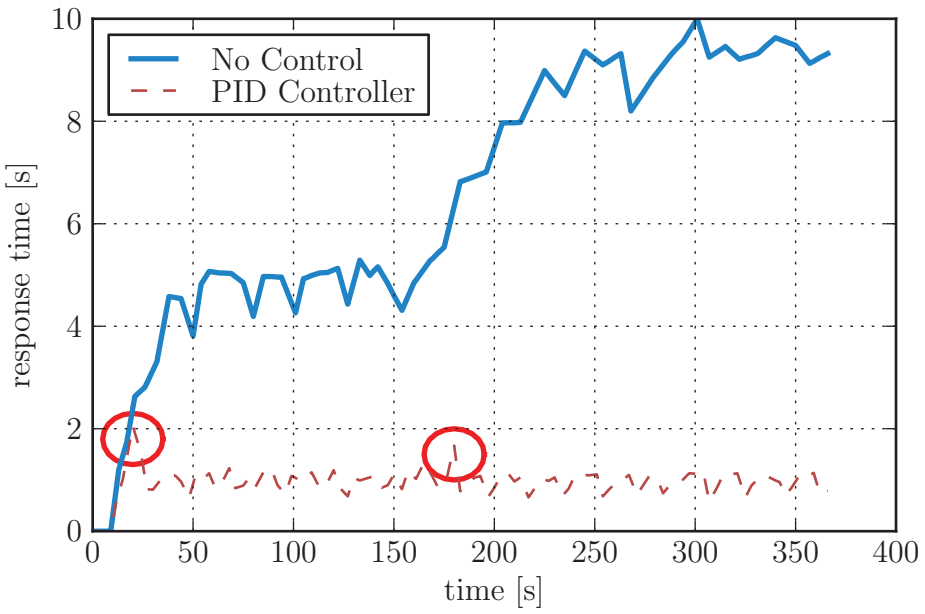


Figure 2.9: Response to system change: Without controller and with PID controller in place

2.3.4 Auction scenario

For the second test we examine the behavior of our controller when used for the characteristic load of our auction scenario. In particular, we examine the performance during the peak load that resembles the bids of a first-price sealed-bid auction [MM87].

To model the peak load we distribute each client's request rate according to a Gaussian distribution with $\sigma = 50$ seconds and $\mu = 130$ seconds. There is a total of 1 000 clients and each client transmits a total of 30 requests. For each request at the server, the server executes a calculation that takes 100 ms. The set-point of the controller that reflects the acceptable processing delay is set to 1 000 ms.

The results of this evaluation for group-based rate control are presented in Figure 2.10. In the first measurement we examine the response time when no rate control is used, i.e., not even open-loop. The maximum response time peaks at around 10 000 ms. The reason for the plateau at 10 000 ms is given by the fact that the concurrency of each client is limited to 1 and that thus for 100 concurrent requests for an operation that takes 100 ms the total average

will not exceed 10 000 ms. In the next three measurements we examine the behavior of our PID controller for different amounts of dead time.

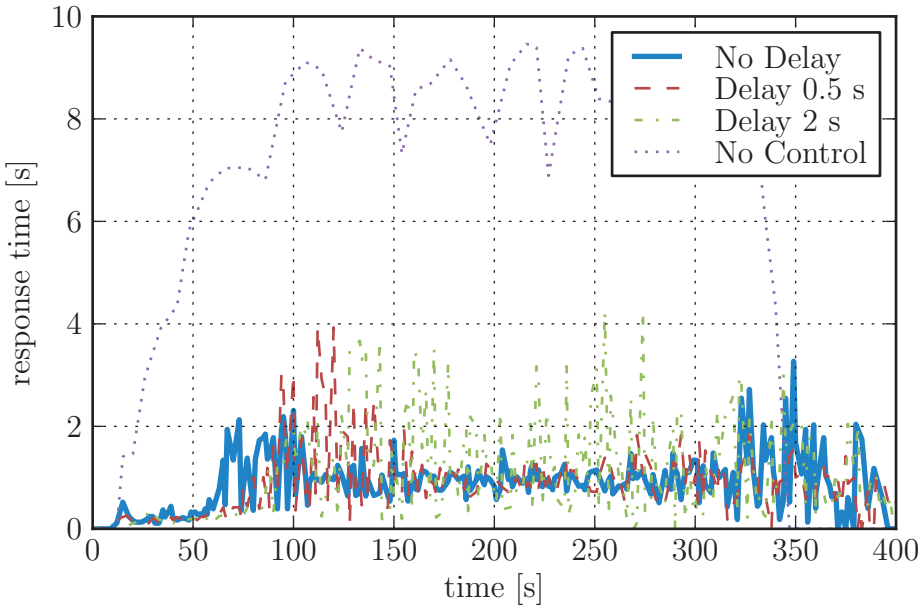


Figure 2.10: Auction scenario with: PID controller, group-based control, average process delay metric

In the first case, the simulated dead time is near zero. The actual dead time is non-zero due to inevitable delays in our broadcast channel. In the second case we simulate a dead time of 500 ms per level of the broadcast tree and in the third case a dead time of 2 000 ms. In each of the cases there is a total of 3 levels in a quinary (5-ary) tree. While the controller is able to constrain the processing delay in the first case, in the second case it takes some time until the system is stable. In the third case we see a relatively large oscillation during the whole test, suggesting that the dead time is too high for our controller. In a real-world application scenario such propagation delays between nodes are considerably lower. If we assume an propagation delay of 50 ms, a total of 30 levels would yield a maximum propagation delay of 1500 ms, comparable to the maximum propagation delay of the simulation with a dead time of 500 ms. A quinary tree with 30 levels would support a theoretical maximum of $1.16 \cdot 10^{21}$ nodes, which is more than anyone would need in practice.

During the evaluation of group-based rate control we observed that the limited granularity of possible group sizes can have a negative impact on the

performance of the system. Especially when large groups are used, the possible group sizes that can be declared using a single divisor and a single modulo value often considerably differ from the manipulated variable calculated by the controller. As mitigation strategy, we enhanced group-based control to use lists of divisors and modulo values, instead of two single values. Clients matching a divisor/modulo combination in the list are allowed to transmit information. This allows for a more fine-grained specification of group-sizes, and thereby for a better stability of the controller.

In Figure 2.11 we show the results for interval based rate control. While the output of interval-based control is stable after the initial peak, variations are slightly larger than with group-based rate control. The stable behavior in Figure 2.11 is achieved by limiting the maximum difference between two consecutive rate control parameters calculated and broadcast by the controller. The respective limit is automatically derived based on the estimated amount of clients and the measured average time per operation. Due to this limitation the controller requires some time until it can reduce the load during steeply increasing client request rates.

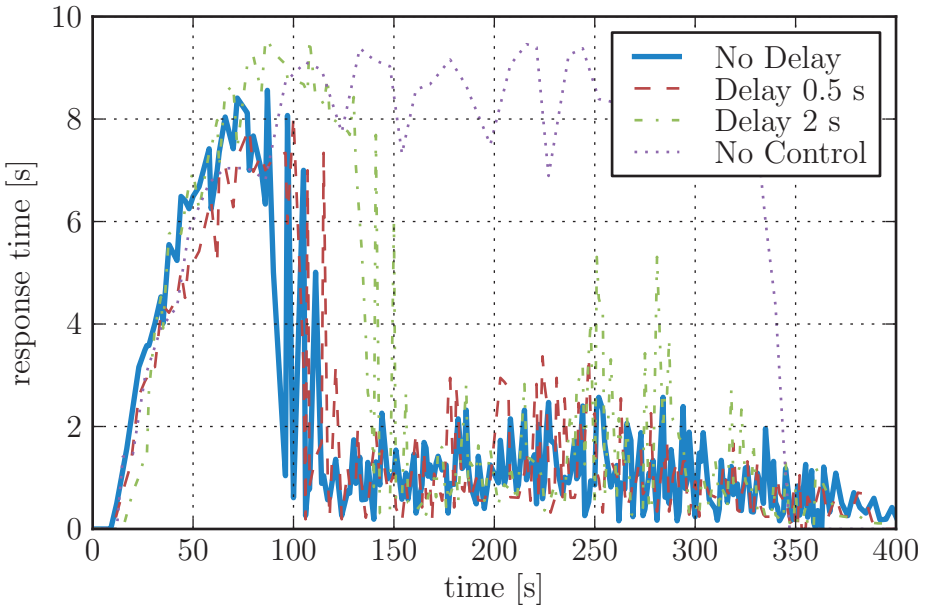


Figure 2.11: Auction scenario with: PID controller, interval-based control, average process delay metric

2.3.5 Comparison

In this section we compare the effectiveness of our approach with two alternative approaches found in related work (Section 2.6) that work without distributed feedback channels.

In the first approach in Figure 2.12 we use a *piggyback* strategy where we embed rate control feedback in HTTP responses. When an HTTP client sends an HTTP request to a server, the server includes rate control information in the HTTP response. The results show that the piggyback approach is unstable and leads to oscillation. A major cause for this problem are variable dead times and the fact that control information at clients cannot be updated between individual requests of a client. Thus, the system is only able to provide request rate control, but not admission control as our distributed feedback channel.

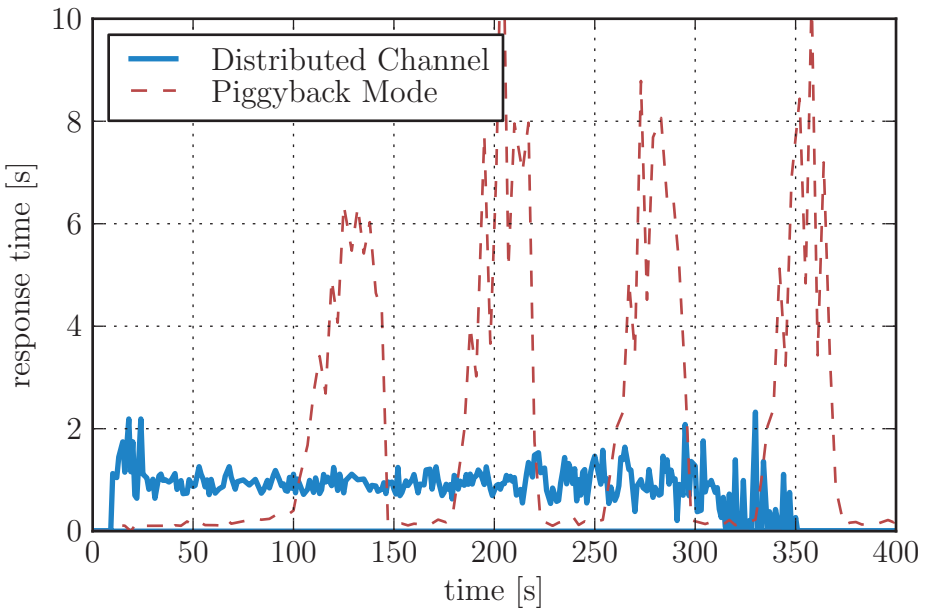


Figure 2.12: Comparison between distributed feedback channel and piggyback approach

In the second approach in Figure 2.13 we use a *reject* strategy similar to the control-theoretic approach by Abdelzaher et al. [ASB02] where we reject requests at the server instead of the client. When a client is not able to establish a connection to the server, we use an exponential backoff strategy similar to TCP's retransmission timer [PA00] without the adjustments using

the SRTT (Smoothed Round Trip Time) [Pos81]. The original approach is not directly applicable to our application scenario, as implementation of a *reject/request-ratio* would not allow us to reject requests on a per-client basis. Furthermore, such an approach would allow unfair clients to gain an advantage by establishing connections with a high frequency. Instead, we reject requests at HTTP level instead of TCP level, as our existing decoupling mechanism depends on the client IDs. As a consequence, the performance of the *reject* approach in our evaluation is restricted by our required adaptations.

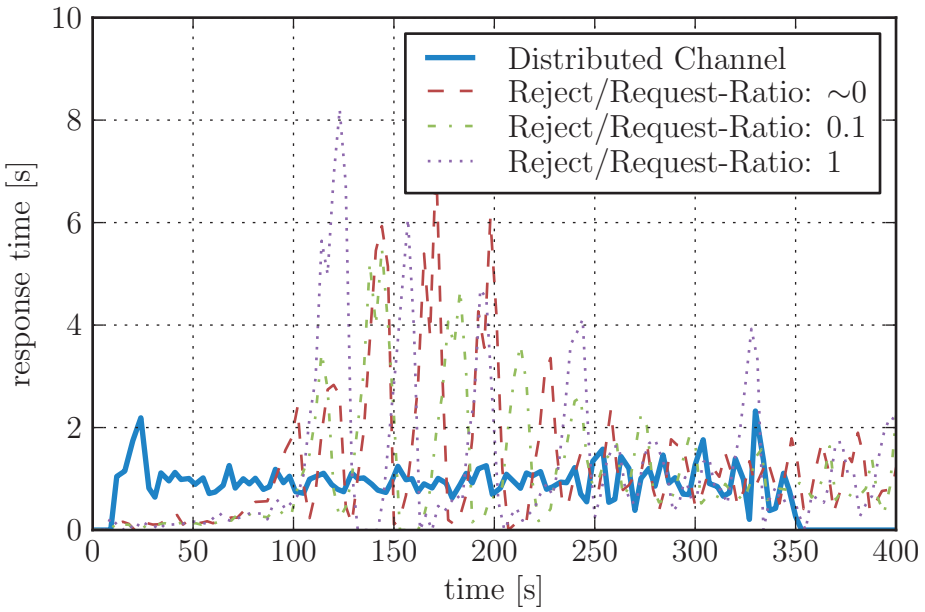


Figure 2.13: Evaluation of different reject/request-ratios

With the *reject* strategy the stability depends on the ratio between the cost of accepting a request and the cost of rejecting a request. Figure 2.13 depicts the results for different ratios between accepting requests and rejecting requests. In the first case processing a request takes 100 ms, while we do not artificially delay the time for rejecting a request. In the additional two cases we increase the time for rejecting a request to 10 ms and 100 ms to simulate a reject/request-ratio of 0.1 and 1 respectively. While the results show that the stability does not significantly differ between different reject/request-ratios, there is a considerable difference in the throughput. While our distributed feedback channel is able to achieve a throughput of 8.6 requests per second, the throughput in the best *reject* approach is only 7.0 requests per second. When increasing the reject/request-ratio the throughput declines further to 5.0 and 2.9 requests per second accordingly. Therefore, the benefits of our distributed

feedback channel are primarily relevant in scenarios with a high reject/request-ratio, where the cost of rejecting a request is not negligible. Abdelzaher et al. have shown that this is the case in Web-server end systems, where they measured a reject overhead of 1.1 ms per request, while processing a request for a zero-sized URL (Uniform Resource Locator) took 1.604 ms [ASB02]. A more detailed examination of the reject/request-ratio's influence is given in the next section.

2.4 Limitations

The applicability of our approach for certain application scenarios can be determined by comparing the effort required at the server for processing requests with the effort required for rejecting requests and with the typical ratio between average load and peak load. For comparison, we assume a simple system that does not use distributed load control: If a request cannot be processed, it is rejected by the system and the client does not try to retransmit the request.

First, we examine the theoretical limitations due to the cost difference between accepting and rejecting requests. We then proceed by discussing the implications of these trade-offs on real-world application scenarios.

2.4.1 Theoretical limitations

We denote the maximum request rate that the system can sustain without rejecting requests as $r_{process}$, the factor between the system load under normal operation and the system load during a peak as $factor_{peak}$, and the factor between the cost of accepting a request and rejecting a request as $factor_{request}$ $(= \frac{cost\ for\ request}{cost\ for\ reject})$. We then calculate an upper bound for the server's effort of rejecting requests when not all requests can be processed by calculating $r_{process} \cdot \frac{factor_{peak}}{factor_{request}}$: We have $factor_{peak}$ as many requests, but rejecting each request only takes $\frac{1}{factor_{request}}$ of the time it would take to process these requests.

If $factor_{peak}$ is equal to $factor_{request}$ the system is not able to handle any business requests during peak load, as rejecting requests takes up all available resources. Thus, in this case our distributed broadcast channel would be able to double the effective capacity of the system, as the resources previously used for load control can now be used for request processing. With a higher $factor_{request}$ the advantage of our distributed broadcast channel decreases,

while with a lower $factor_{request}$ the advantage increases. For example, if $factor_{request}$ is 100, but $factor_{peak}$ is only 10, the upper bound of rejection costs during the peak load is about $\frac{1}{10}$ of the system's overall performance. In this case, the advantage of *outsourcing* the broadcast channel is rather limited.

2.4.2 Real-world applications

In a real world system the examination of our system's advantages are more complex than the calculations for the simplified model given in Section 2.4.1: (i) Our model does not deal with retransmission of requests. If a request is rejected, the client would need to try a retransmission at some point. This increases the advantage of our approach. (ii) We assume that each request is transmitted independently of other requests. In reality, a client can use information from earlier requests to optimize the rate for later requests. This in turn decreases the advantage of our approach, but may lead to unstable behavior as shown in Figure 2.12.

While our approach cannot prevent clients from *trying* to cheat by ignoring control information, the server can easily verify if individual clients act according to the broadcast control information. If clients continuously violate those rules, the server can ban those clients from further participation in the auction by rejecting requests without processing them. In addition, to prevent clients from injecting bogus messages into the network, clients will only forward messages digitally signed by the server.

2.5 Security vs. dependability

As discussed in Section 2.2 there is a security vs. dependability trade-off given by the maximum allowed delay for a bid. A small delay increases security, as this delay corresponds to the amount of time by which an attacker can theoretically backdate a bid. However, a small delay also decreases dependability, as in the case of transmission problems clients have less time to resubmit a bid. There are two extremes between which the value for the maximum allowed delay can be set:

- From a security point of view the optimal maximum allowed delay is zero. Such a value effectively disables decoupling, as the allowed

timestamp of a bid must always correspond to the time when the bid is received at the server.

- From a dependability point of view the optimal maximum allowed delay is not limited. However, for practical reasons a limit is required to evaluate the outcome of the auction.

Those two extremes also influence the strategies used for bid transmission. For example, if bid overwriting is used, in case of the second extreme there is no motivation for a client to submit any but the final bid, as transmission of prior bids does not influence the outcome of the auction. The ultimate goal of our decoupling approach is to allow for both—adequate security and adequate dependability. We facilitate this goal via the following steps:

- First, we extend our decoupling strategies to not only limit the transmission rate of clients, but to also control the maximum allowed delay for a bid. The maximum allowed delay is given as offset to the smallest allowed delay specified by the decoupling strategy. For example, if a decoupling strategy mandates that the next bid can be transmitted in one minute from now, we can specify a maximum allowed delay of ten seconds, implying that the bid must be transmitted within one minute and ten seconds from now.

- Second, we allow to adaptively increase the maximum allowed delay in case of failures. For example, if the auctioneer observes overload of the network or server infrastructure, bids from bidders can be accepted even though they do not arrive in time.

Concluding, the goal of our decoupling strategies is not only to facilitate an optimal server load, but also to keep transmission delays at the individual clients' low, as this allows to decrease the range of possible time values that an adversary can assign to a bid.

2.6 Related work

Our adaptive rate control approach is not the first effort to dynamically control request rates of clients on higher level application layers. For example, several papers [ASB02, LLA+02, PPBG09] describe approaches that use controllers to react to system load and thereby manage to improve the system's overall

performance. The main difference to our work is that we do not take actions on the server side (such as rejecting requests or content adaption), but rather try to solve the problem directly at the source—at the client side. This section gives a brief overview of relevant and related work. First, we discuss load control mechanisms in standard Internet protocols. Afterwards, we proceed to specialized solutions for particular types of Internet and Web applications.

TCP based servers [Pos81] are able to implicitly control the transmission rate of clients by only allowing a particular amount of parallel connections and by using a *window* to restrict the amount of data waiting to be processed. If SYN packets of clients are dropped, clients use an exponential backoff approach [PA00, MK08] to time retransmissions, which can further be improved with a PI controller [LY09]. In our application scenario, TCP based rate control is not fully applicable as our goal is to prevent overload by filtering requests directly at the clients.

The goal of RacingSnail [GSTBU10] is to monitor and optimize the performance of existing systems. The authors state that each system exhibits an optimal performance at a specific request rate. With a lower request rate the system is underloaded, and with a higher request rate the system is overloaded. RacingSnail is implemented using a blackbox approach where one module is responsible for measuring and analyzing the server's performance, and another module is responsible for slowing down client request rates. Compared to our adaptive rate control approach the application scenario and the solution approach are different. While our system deals with scalability aspects in case of large amounts of clients distributed over the Internet, the application scenario of RacingSnail deals with smaller amounts of clients and thus does not require a scalable feedback channel. As a consequence, our system mostly contributes with techniques required to facilitate request rate control for large amounts of clients, while RacingSnail deals with issues such as how existing blackbox services can be slowed down.

A control-theoretical approach for performance guarantees for Web server end-systems [ASB02] by Abdelzaher et al. describes performance control of a Web server using classical feedback control theory, thereby providing overload protection, performance guarantees, and service differentiation in the presence of unpredictable loads. The goals of the authors are performance isolation (to isolate virtual hosts from each other), service differentiation (to give more priority to more important clients), and QoS (Quality of Service) content adaption (to adapt the content according to the designated quality of service, e.g., by serving pictures with a lower quality in case of high server load). The main difference to our approach is that we filter requests before they are

transmitted to the server, instead of rejecting requests at the server. Rejecting requests on the TCP layer is not directly applicable to our protocol, as we first need to identify a client to determine if a request should be rejected. In addition, implementing a reject/request-ratio for all clients is not meaningful in our application scenario, as such a ratio would allow unfair clients to gain an advantage by establishing connections with a high frequency.

Q-PID, another control-theoretic approach for Web services has been suggested by Lim et al. in a paper that describes adaptive admission control for a cluster-based Web server system [LLA+02]. It uses a controller based on a feedback method. By averaging the response times of previous requests, the controller calculates an *accept ratio* that specifies which percentage of incoming requests should be rejected. Thus, the system is able to guarantee bounded and predictive response times by the Web service. As in the case of Abdelzaher's approach the main difference to our contribution is that we directly control the request rates of clients, while the Q-PID approach rejects requests at the server.

Chan et al. propose a fuzzy PI controller to guarantee proportional delay differentiation on Web servers [CC07]. They argue that it is not economically feasible to design Web servers for peak load, and that even such a design does not protect Web servers from overload. Additionally, they identify that during peaks not all requests can be served in a timely manner. Consequently, their solution approach is to use a fuzzy PI controller to provide a better service to a premium class of users. Such an approach is not directly applicable to our application scenario, where all users should have the same conditions for submitting bids. However, the general idea could complement our solution, if we specify a premium class of requests, instead of a premium class of users. This would allow us to provide a better service for more important types of requests when the server is overloaded.

Philippe et al. describe an approach for a self-adapting service level [PPBG09] that allows some components in an application server to dynamically degrade or upgrade their level of service, thereby trading a lower service level for a better overall performance of the server. This could complement our solution in general, but is not applicable for our specific application scenario.

In addition to the discussed publications there is a number of publications [MOZ+09, KRAW08, SRS00, VG02a, VG02b] that deal with admission control for Web server systems. Most existing systems adapt parameters at the server to influence the load produced by clients, e.g., by deciding which requests should be rejected. In contrast, our technique frees the server from rejecting individual requests, as we directly filter requests at the client.

2.7 Conclusions

This chapter presents a new approach for adaptive load control and performance in our application scenario of first-price sealed-bid auctions. Our main contribution is the integration of (i) a distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain client requests directly at the client side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization.

In comparison to established techniques, our system allows to control the quality of higher layer protocols without relying only on the load control mechanisms provided by lower layer protocols. Unlike related work, the servers in our system do not need to communicate with each single client individually. In particular, we can control request rates of clients even before a connection to a server is established, leading to a significant reduction in the required server-side resources. Our system does not only prevent clients from overwhelming the capacity of the servers, but also allows to reduce the capacity required at the server-side infrastructure for applications that show temporary peaks in transmission rate or that can queue and send client requests in a single batch request, whereby data obsoleted by newer information can already be filtered at the client side.

Concluding, our approach provides viable means to manage high loads in first-price sealed-bid auctions without requiring large clusters able to cope with brief peak loads. Controlling transmission rates at clients decreases the number of required servers and makes more efficient use of available resources. A potential drawback is the necessity of a distributed feedback channel, which, however, could be approximated through similar infrastructures.

Chapter 3

Interval-based timestamping

This is the first of two chapters in the area of time synchronization and timestamping. In particular, our contributions in this chapter are: (i) A timestamping protocol and implementation capable of operation on smart cards, (ii) an examination of the restrictions placed by common smart card environments, and (iii) an experimental evaluation of the time synchronization capabilities of different smart card environments.

In our application scenario interval-based time synchronization and timestamping is required for the secure timestamping of bids that are not immediately transferred to the auction server. When synchronizing the time our interval-based protocol is less susceptible to network delays caused by adversaries, as such delays are directly reflected in the interval size. In addition, our smart card implementation is essential for security: Secure timestamping cannot be performed on the bidder's computer, as this would make it trivial for bidders to manipulate timestamps.

In Section 3.1 we define the problem, followed by Section 3.2 where we contribute with our interval-based time synchronization protocol. Based on our smart card implementations we then prove the feasibility of our approach in Section 3.3. Afterwards, Section 3.4 discusses related work. Finally, we draw our conclusions in Section 3.5.

3.1 Problem definition

To motivate our interval-based time synchronization protocol this section examines the means of an attacker to tamper with time synchronization requests when traditional time synchronization protocols are used.

NTP (Network Time Protocol) [MMBK10] is a time synchronization protocol used on the Internet. To obtain timestamps from a remote server, NTP records four timestamps ($T_1 \ldots T_4$) as depicted in Figure 3.1. T_1 and T_4 are assigned according to the client's clock, while T_2 and T_3 are assigned according to the server's clock. NTP then uses the formula $\frac{(T_2-T_1)-(T_4-T_3)}{2}$ to determine the offset between the client's and the server's clock. Using the offset, it tries to estimate the skew of the local clock in relation to the server's clock and then gradually corrects the value and the frequency of the clock using a software-based PLL (Phase-Locked-Loop) implementation.

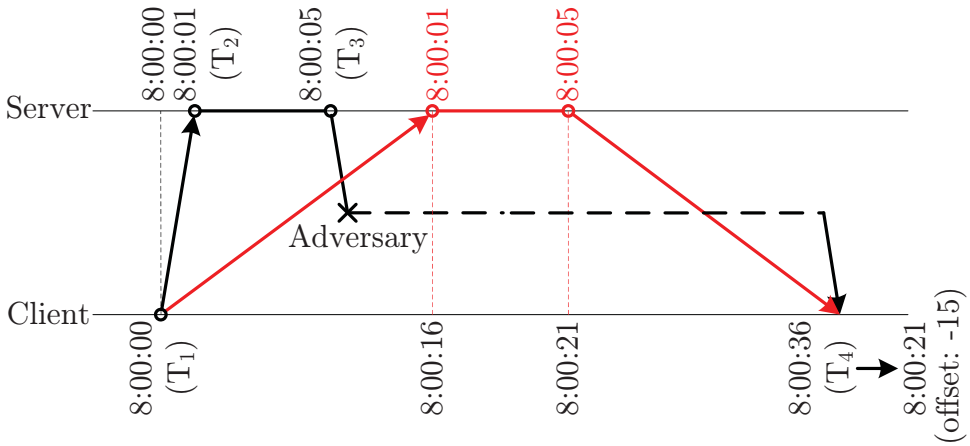


Figure 3.1: Asymmetric propagation delay

Figure 3.1 shows an example of how an adversary can use asymmetric propagation delays to affect time synchronization when a time synchronization protocol assumes symmetric propagation delays and uses the offset formula used by NTP. Initially, client and server are perfectly synchronized. The client sends its request at 8:00:00, which is received by the server at 8:00:01. The server processes the request and sends the response at 8:00:05. On the way back to the client, the response is delayed by an adversary, causing the client to receive the delayed response at 8:00:36. When the client calculates the offset to the server, this yields a value of -15 seconds, implying the time 8:00:36 on the client's clock corresponds to the time 8:00:21 at the server's

clock. Therefore, the adversary delaying the message on the way back to the client eventually delays the client's clock by 15 seconds.

3.2 Time synchronization and timestamps

To mitigate the problems described in the previous section we use an interval based approach based on Marzullo's work [MO85], allowing us to represent time as a correctness interval that represents the inaccuracies due to network delays and clock characteristics, such as clock skew and limited precision. An interval representing a particular time is correct, if the represented time is within the bounds of the interval. Two intervals are *compatible* if they represent the same time value, such as "5 pm" or "now". Two intervals are *consistent* if their intersection is not zero. Thus, in the case where two compatible intervals are not consistent, at least one of these intervals must be incorrect. In our algorithm we use *offset intervals* that represent the difference between the smart card's local clock value and the values of the two intervals endpoints.

Both, our time synchronization and timestamping algorithms are adapted to smart card environments that may exhibit high propagation delays between smart card and timeserver. In particular, our contributions over the state of the art are:

- An adaption of Marzullo's interval approach to guarantee that each time synchronization step—independent of the propagation delay—only increases, but never decreases, the accuracy of the local clock.
- An adaption of Marzullo's intersection approach to a single time server scenario, allowing us to detect if an attacker has been able to successfully tamper with the time on the smart card.
- A roundtrip delay mitigation approach that allows use to decrease the size of intervals applied as timestamps, if there are confirmed problems at the auctioneers infrastructure that lead to consistent, measurable delays in the processing of time stamps.

The definitions used in this section are based on *Network Time Protocol Version 4 – Reference and Implementation Guide* [Mil06b] by David L. Mills and on *Maintaining the time in a distributed system* [MO85] by Marzullo and Owicki.

3.2.1 Time synchronization

We define our time synchronization protocol based on algorithms implementing the different functions: Algorithm 1 illustrates the main control loop in the client. The time transfer from the server to the client is done by Algorithm 2 with Algorithm 3 being responsible for increasing the interval's size due to clock skew. Both algorithms work with offset intervals that represent the uncertainty in clock synchronization due to clock skew and network delays, for example. As these offset intervals do not represent a real point in time, Algorithm 4 converts the offset intervals to absolute time intervals that are then cryptographically signed in Algorithm 5. The interpretation of the timestamped intervals is the server's task and illustrated in Algorithm 6.

Time synchronization protocol

Algorithm 1 shows the time synchronization algorithm running on the client. First, the algorithm initializes the offset interval i_{local} to $[-\infty, \infty]$—as we do not have any information about the current time yet. It proceeds, by obtaining an interval from the timeserver that is subsequently stored in i_{server}. The local clock value returned by $get\_time()$ when the time synchronization step was started is stored in t_{last_set}.

If the two compatible intervals i_{local} and i_{server} representing the current time do not overlap, we can conclude that one of these intervals must be incorrect. In this case this represents a fatal error and we disable the smart card, requiring the bidder to use other means to submit bids. As the inverse condition does not need to be true (two compatible intervals may intersect if one or both of these intervals are incorrect), this test does not detect all types of failures. However, the maximum error in these cases can be considered rather low, as it cannot be larger then the smallest value of i_{server} obtained during all preceding time synchronization steps.

When assuming that both i_{local} and i_{server} are correct, we can follow that the *intersection* of i_{local} and i_{server} will also represent the correct time. Therefore, we replace i_{local} with the *intersection* of i_{local} and i_{server} to decrease the size of the resulting interval and to more accurately represent the current time. An example is given in Figure 3.2.

Finally, the algorithm waits for the next time synchronization step while allowing the smart card to timestamp bids in between. Before executing the next round, we increase the size of i_{local} with the $extend()$ function, to account for inaccuracies due to clock skew.

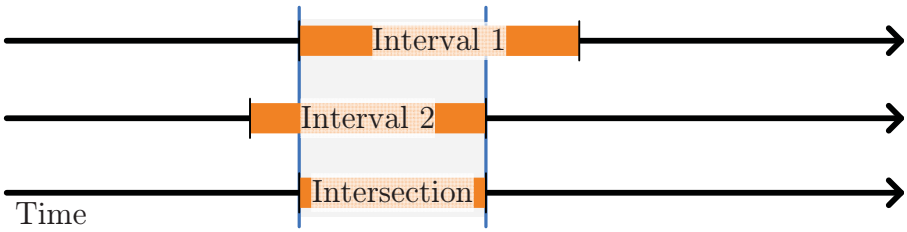


Figure 3.2: Intersection between two intervals

Algorithm 1 Obtaining time from server

$i_{local} \leftarrow [-\infty, \infty]$
loop
  $t_{last\_set} \leftarrow get\_time()$
  $i_{server} \leftarrow recv\_interval()$
  if $intersection(i_{local}, i_{server}) == 0$ **then**
    $abort\_with\_error()$
  end if
  $i_{local} = intersection(i_{local}, i_{server})$
  while $wait\_for\_timeout()$ **do**
    {Apply timestamps on incoming bids}
  end while
  $i_{local} = extend(i_{local})$
end loop

Time transfer from server to client

The $recv\_interval()$ function given in Algorithm 2 is used to obtain the time from a remote server: After receiving the response by the server, the smart card first calculates the server's synchronization distance (Γ) from the server's root dispersion (E) and the server's root delay (Δ), which are both part of the NTP response message [MMBK10]. The synchronization distance represents the maximum error in the server's response due to all causes. Afterwards two offsets are calculated, by decreasing and increasing the obtained offsets by half the roundtrip delay of the time synchronization request. Finally, the offset values are combined with the synchronization distance and returned as a single offset interval.

Algorithm 2 *recv_interval*(): Time synchronization request

$response \leftarrow receive\_response\_from\_server()$
$\Gamma \leftarrow response.E + \frac{response.\Delta}{2}$
$o_1 \leftarrow (T_3 - T_4)\ \{(T_3 - T_4) == offset - \frac{roundtrip\_time}{2}\}$
$o_2 \leftarrow (T_2 - T_1)\ \{(T_2 - T_1) == offset + \frac{roundtrip\_time}{2}\}$
$i_{server} \leftarrow [o_1 - \Gamma, o_2 + \Gamma]$
return i_{server}

Clock skew

The *extend*() function in Algorithm 3 increases an interval to account for the effects of clock skew. ρ_K represents the precision of the remote clock and is included as part of the time server's response. ρ represents the precision of the local clock and is estimated by the auctioneer. $\Phi(X)$ represents the maximum error due to clock skew during a period of length X. In our application we are interested in the clock skew since the last time synchronization step t_{last_set}. *extend*() combines the different effects of these components, and increases the size of the interval accordingly.

Algorithm 3 *extend*(i): Transform offset interval due to effects such as clock skew

$\varepsilon \leftarrow \rho_K + \rho + \Phi(get\_time() - t_{last\_set})$
$i_{new} \leftarrow [i.left\_offset - \varepsilon, i.right\_offset + \varepsilon]$
return i_{new}

3.2.2 Timestamping

When timestamping a bid we first convert the offset intervals i_{local} and i_{server} to absolute intervals with the approach shown in Algorithm 4. Afterwards, we append these absolute intervals plus a sequence number to the bid, and sign the resulting data with a digital signature as shown in Algorithm 5.

Algorithm 4 *abstime*(i): Transform offset interval to *current* time

$t_{current} \leftarrow get\_time()$
$i \leftarrow extend(i)$
$i_{abs} \leftarrow [t_{current} + i.left\_offset, t_{current} + i.right\_offset]$
return i_{abs}

Algorithm 5 $timestamp(bid, i_{local}, i_{server}, t_{local})$: Timestamp incoming bid

$seq \leftarrow seq + 1$
$ts_{local} \leftarrow abstime(i_{local})$
$ts_{server} \leftarrow abstime(i_{server})$
$data \leftarrow concat(bid, seq, ts_{local}, ts_{server}, get\_time())$
$sign(data)$

While it would be sufficient to only include the absolute interval derived from i_{local} in the timestamp, the interval derived from i_{server} allows the auctioneer the mitigation of high peak loads as described later on.

A sequence number is required to verify that all bids signed by the smart card have been received at the server. After the auction ended, we therefore require the smart card to transfer the value of its sequence number to the server.

Server-side timestamp interpretation

Algorithm 6 shows how timestamps are interpreted at the server. First, it is checked if there have been confirmed problems at the auctioneer during the time represented by the timestamp. If this is the case, we can use the $adapt()$ function, to decrease the size of ts_{server} accordingly.

As example, consider that the auctioneer's monitoring infrastructure confirms that there has been an additional network delay of 10 seconds for messages that travel from the network to the auctioneer. This would result in the right hand side of each interval obtained over the network during this delay—given as $(T_2 - T_1)$—to be increased by 10 seconds. Therefore, our $adapt()$ function can shift the right hand side of the interval 10 seconds to the left, to mitigate for the increased propagation delay.

Finally, the algorithm calculates the actual timestamp by building the intersection of ts_{local} and ts_{server}. While under normal conditions ts_{local} is already a subset or equal to ts_{server}, in cases where ts_{server} was recalculated with $adapt()$, the intersection potentially allows to reduce the size of the interval.

Decision if a bid should be accepted

Our application scenario requires that any bid timestamped before the auction deadline must be accepted, while any bid timestamped after the deadline must be rejected. In cases where both interval endpoints are either before or

Algorithm 6 Timestamp processing at server

if *signature incorrect* **then**
  $abort\_with\_error()$
else if $ts_{server} \subseteq t_{confirmed\_problems\_at\_auctioneer}$ **then**
  $ts_{server} \leftarrow adapt(ts_{server})$
end if
$ts_{timestamp} \leftarrow intersection(ts_{local}, ts_{server})$

after the deadline, the decision is clear. However, in cases where the interval overlaps with the deadline as shown in Figure 3.3, the auctioneer cannot fully assert, if a bid was placed before, or after the deadline.

Figure 3.3: Interval overlaps with deadline

In most cases, intervals of timestamps are relatively small. The only case when a large interval will be assigned is if there has not been any single time synchronization step with a reasonable roundtrip delay and—additionally—if the auctioneer did not detect any overload at his local network, and thus does not call *adapt*() on the assigned timestamp. In such cases it can be possible that large intervals are caused by delay attacks caused by malicious bidders.

To mitigate for potential attacks, we always compare the right hand side of an interval—which represents the latest possible time of a bid—to the deadline. In cases where intervals are small, it does not matter which part of the interval is compared with the deadline, as the expected interval size is at most a few hundred milliseconds. However, in cases of large intervals—e.g., due to delay attacks—comparing the right hand side to the deadline prevents these large intervals from delaying the auction deadline for a particular user.

The worst case from the bidders' perspective occurs if every single time synchronization step features high propagation delays to the server, while there are no confirmed problems at the auctioneers infrastructure. In such cases, the deadline comparison with the interval's right endpoint advances the bidders' deadline, and thereby requires bidders to submit bids sometime before the auction deadline. However, as bidders can always query the smart card's time, they can detect such cases in advance, allowing them to also use other mitigation strategies, such as switching to a different network connection.

Figure 3.4 shows how this approach affects the accuracy of time stamps with measurements based on our protocol implementation. The example simulates a delay attack with high propagation delays for responses sent from the server. The "reference clock" is regarded as the server's time, while the "offset against reference clock" is the right hand endpoint of the interval stored on the smart card. In the example the effects of clock skew are negligible and therefore not visible.

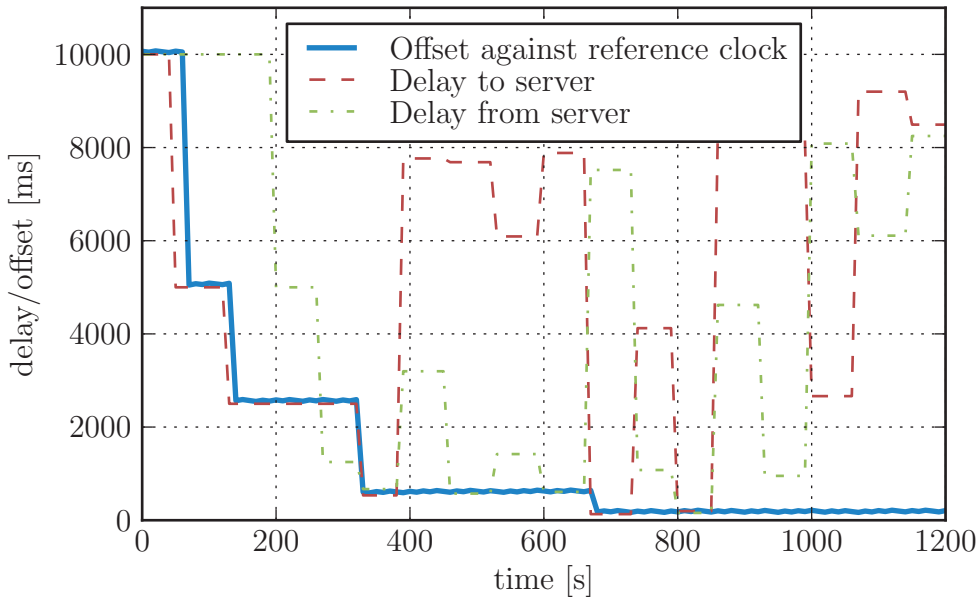


Figure 3.4: Response to delay attack

Initially, the "offset against reference clock" is equal to the propagation delay of the first time synchronization message sent from the client to the server, as the formula for the right endpoint $(T_2 - T_1)$ is only dependent on the propagation delay to the server, but not on the propagation delay of responses sent by the server.

Subsequent time synchronization steps will calculate the intersection of the smart card's local offset interval and the offset interval obtained from the time server. Therefore, with each time synchronization step the right endpoint can either stay equal or move to the left, but never to the right. This behavior can be seen in Figure 3.4, as the "offset against reference clock" only decreases but never increases as clock skew is negligible.

In cases where the smart card's clock skew would be noticeable, the "offset against reference clock" would slowly increase over time, but still be reset by

each time synchronization step, where the propagation delay to the server is lower than the offset.

3.3 Smart card implementation

In this section we examine techniques to implement our approach on .NET cards and Java cards. While our general approach is also applicable to other types of secure devices, such as TPMs (Trusted Platform Modules), our original application scenario, and thereby also our evaluation, targets smart card environments.

To implement our time synchronization protocol on a smart card, the smart card needs to possess the following capabilities:

1. Access to an internal timer clocked by an oscillator internal to the smart card. The frequency of the oscillator must not depend on the frequency provided by the card reader to the `CLK` contact.
2. A means to detect if the counter has been reset, e.g., because the power supply to the smart card was interrupted.
3. The ability to detect overflows of the counter. However, as explained later on, a software based workaround is possible, if this feature is missing.

A read-only timer is sufficient, as the difference between the reference time and the timer's value can be stored as a variable on the smart card. Furthermore, we do not presume a battery powered clock that allows keeping the time while the secure device is not connected to a computer, as this feature is not available on a majority of examined devices.

The NTP packet format is a potential cause for compatibility issues as smart cards need to successfully generate and parse the individual fields of packets. This parsing cannot be outsourced to external applications as smart cards must be able to verify signatures applied by the time server. If the values would be extracted by outside applications, smart cards would have no means of verifying that the extracted values actually correspond to the cryptographically signed response of an NTP server.

NTP messages are parsed and generated on-card. The smart card communicates via APDUs with a client application running on the local PC,

relaying NTP messages between smart card and Internet. We use MD5 (Message-Digest algorithm 5) based signatures as described in [Mil06a] with an individual key for each smart card, requiring smart cards to provide an MD5 hash function to on-card applications. While collisions in MD5 have been found [WY05], control over both, NTP client and server implementations, allows us to increase the security beyond the NTP specification by replacing MD5 with any alternative hash function.

For the evaluation discussed below we implemented our protocol on a *Gemalto .NET IM V2* smart card and—additionally—tested the capabilities of a technology preview release (simulator) of the connected edition for Java card 3 and evaluated the limitations of Java card 2 based on a *NXP (NXP Semiconductors) JCOP (Java Card OpenPlatform) 41 V2.2.1/72k* card. All three types of cards are programmable and allow the installation of appropriate on-card applications.

3.3.1 .NET card

The .NET smart card API (Application Programming Interface) provides access to the 32-bit read-only `TickCount` property in `System.Environment` that returns the the number of milliseconds passed since the smart card has been powered on. The timer within .NET cards runs continuously for 24.9 days until it reaches `Int32.MaxValue`. Afterwards, it wraps back to zero. According to the API, the resolution of the `TickCount` property is at least 500 milliseconds. Similar to `currentTimeMillis()` in Java card 3, the `TickCount` property can be used to measure the time between two invocations.

The `TickCount` property overflows after 24.9 days, making it impossible for the software running on the smart card to assess whether, e.g., 25.0 days or 0.1 days have passed. This is mostly a theoretical issue, as the usual duration of auctions is much shorter in our application scenario. Detection of overflows would be possible if the software running on the smart card could check the value of the `TickCount` property in intervals smaller than 24.9 days. However, the .NET API does not allow for the execution of background threads. Hence, code execution depends on external APDUs transmitted to the smart card. As these APDUs need to be transmitted over the bidders computer, it cannot be guaranteed that the bidder abides by a policy of sending APDUs with intervals smaller than 24.9 days.

To work around this problem, the auctioneer can limit the maximum time allowed between the application of a timestamp and the transmission to the auction system to an interval smaller than the maximum value that can

be represented by the timer. While this does not prevent the timer from overflowing, it allows the auction system to detect if an overflow occurred.

Concluding, the .NET card fulfills our requirements with the overflow restriction described above.

3.3.2 Java card 3

In Java card 3 the *Application Programming Interface for the Java card Platform, Connected Edition* provides a `System.currentTimeMillis()` method that returns the current time in milliseconds. While the Java card API also provides a method `synchronizeTime()` that allows obtaining the time from an external time reference, the API does not specify what external time reference is used by implementations. Therefore, we cannot trust that the time obtained by a Java card 3 is valid, as the external time reference can be under control of an adversary, e.g., if the time is obtained from an external card reader. As a consequence, our only assumption is that the difference in the responses of two `System.currentTimeMillis()` invocation reflects the interval in milliseconds between these invocations.

A further complication is that some future Java card 3 implementations might automatically obtain the time from an external reference. In order to detect such events, Java card 3 allows to register a event handler for `event:///platform/clock/resynced` events. Such events are fired after the Java card obtained its time from an external reference. The time delta between the old time and the new time is included in the arguments of the event handler. Capturing the events allows a time synchronization protocol to mitigate the offsets caused by such synchronization steps.

While Java card version 3 has been standardized, there are currently no cards available on the market. Consequently, Java card 3 only represents an option for the future.

3.3.3 Java card 2

The official Java card 2 API does not include methods to access the timers of smart cards. However, there are implementations of Java card 2 compatible devices that include access to an internal clock, such as the DS1955B and DS1957B cryptographic iButtons produced by Dallas Semiconductor (now a subsidiary of Maxim Integrated Products). The internal clock of these devices is quartz-driven and powered by an internal battery. However, according to

information we received by a Maxim engineer, Java-powered iButtons have been discontinued.

As a consequence, we based our Java card 2 prototype version on standard Java cards. While the API of these cards does not provide access to a timer, we used a software based timer simulation for evaluation purposes, by regularly updating the value of an internal timer variable with incoming APDUs and by providing a corresponding access function that guarantees a strictly monotonic increasing behavior.

In our implementation it was not possible to provide an interface compatible to `System.currentTimeMillis()`, as `currentTimeMillis()` returns a 64-bit *long* value, while the largest data type supported by Java card 2 is (depending on the card) either signed 16-bit or signed 32-bit. However, Java card version 2.2.2 provides a `BigNumber` class that allows using multiple smaller data types to simulate a larger data type. As, we could not obtain Java cards implementing the relatively recent Java card 2.2.2 specification, we implemented our own `BigNumber`-like class based on a subset of the `BigInteger` class in Java SE (Java Platform, Standard Edition).

In a first preliminary performance evaluation we tested a simple application that sums all integers between 10 000 and 20 000. On a JCOP card this test shows about 200 operations per second. In comparison, a .NET card natively implementing the required data types is able to execute about 5 800 operations per second, nearly a thirty-fold difference.

To sum up, Java cards version 2 are not adequate for time synchronization. They miss standardized access to a timer as well as appropriate data types to process time synchronization packets.

3.4 Related work

While secure devices and time synchronization protocols are both well researched topics, the combination of using a secure time synchronization protocol in trusted hardware devices under physical control of potential adversaries has not been adequately researched in the academic area. In this section we therefore provide related work in the area of time synchronization. First we discuss generic work, and later on concrete time synchronization protocols.

Lundelius and Lynch proved that the optimum bound achievable when synchronizing n clocks is $u \times \left(1 - \frac{1}{n}\right)$, where u is the uncertainty in message

transmission time and n is the number of hosts [LL84]. In a traditional client-server setup where a single client obtains its time from a single server, the maximum message transmission delay between client and server limits the uncertainty, and thereby the possibilities of an attacker to $\frac{u}{2}$. In terms of the four timestamps T_1–T_4 we can define the maximum error as $\frac{(T_4-T_1)-(T_3-T_2)}{2}$.

Interval based time synchronization [MO83] as proposed by Marzullo and Owicki is an alternative view on time synchronization. Interval based protocols account for the fact that time synchronization with perfect precision and accuracy is not possible and therefore try to determine an interval that contains the real time, instead of a single point that represents the real time. An accuracy interval represents the fuzziness in setting, keeping and reading the time of a clock. For a perfect clock the length of such an accuracy interval is zero. In the real world, the effects of clock skew and network delays lead to an accuracy interval with a length larger than zero. With increasing time (without time synchronization) the size of the accuracy interval increases because of inaccuracies of the clock.

A security analysis of NTP protocol version 2 was conducted in 1990 by M. Bishop [Bis90]. While the evaluated NTP version is dated, the analyzed attacks, such as replay attacks, delay attacks, and denial-of-service attacks, are still relevant to today's time synchronization protocols.

RADclock (Robust Absolute and Difference Clock) [RV10, VRK09] is a project at the University of Melbourne which is developing replacements for NTP clients and servers. As our interval-based approach, RADclock is designed to be robust against asynchronous and variable delays on the network. Unlike NTP, which uses a feedback design, RADclock uses a feed-forward design. The main advantage of such a design is that it can mitigate the problems of a PLL (Phase-Locked-Loop) and FLL (Frequency-Locked-Loop) in case of high variable delays. In particular, there are two main disadvantages of PLL and FLL approaches in comparison to feed-forward designs [RV10]: First, the stability of such approaches cannot be guaranteed and if they lose their lock this can result in shifts to high-error modes. Second, it is not possible to define difference clocks—which provide higher accuracy and higher robustness than absolute clocks—in a feedback framework. RADclock obtains the time in two distinct steps: (i) It uses rate synchronization to bootstrap a clock that can act as a difference clock, and (ii) it uses the synchronized difference clock to assess the round-trip delay during synchronization of an absolute clock. Compared to RADclock, our protocol does not have a notion of quality. Instead, our interval-approach automatically prefers time synchronization steps with low round-trip delays. In addition, our interval based approach is

able to combine information from several time synchronization steps, in order to reduce the combined interval size.

3.5 Conclusions

In this chapter we contributed with a secure time synchronization and timestamping approach for our first-price sealed-bid auction application scenario. In traditional timestamping protocols such as NTP, adversaries can use network delays to tamper with the results of time synchronization. Depending on the exact nature of those delays this either allows to advance a clock, or to delay a clock. In our interval-based timestamping protocol we mitigate this problem by using interval sizes that reflect network delays during time synchronization. Therefore, the auctioneer is able to detect possible attacks and to treat them accordingly.

Furthermore, even in case of systematic inaccuracies that cannot be mitigated by our interval-based approach our protocol shows benefits in comparison to a traditional auction system, where delays are not known beforehand:

- In our protocol systematic inaccuracies can decrease the accuracy of a timestamp, thereby forcing the user to place the bid already some time before the deadline. However, we can detect such cases and inform the bidder before a bid is placed.
- In a traditional auction scenario network latency delays the time of bid placement. However, the user does only learn that such latencies have occurred after the bid has been transmitted to the server. In contrast, in our approach the user can learn about latencies before submitting a bid. Therefore, the user can take mitigation strategies, such as placing a bid already some time before the auction deadline or using an alternate trusted time source.
- Finally, continuous high network latencies lead to larger time intervals at the client. In case of problems at the auctioneer's infrastructure, the auctioneer can detect and mitigate these problems. First the auctioneer needs to measure the transmission and processing delay occurring at the auction server and the network on which the auction server is located. In the second step the auctioneer can then retroactively decrease the interval sizes assigned to timestamps, if such high latencies occur over longer durations.

In addition, we implemented our protocol on different types of smart cards and provided an evaluation of the results. The smart card implementation secures the protocol against attacks by malicious bidders, as those bidders cannot manipulate the software running inside the smart card. Our evaluation shows that .NET cards are currently the most feasible option for the implementation of our protocol. However, once Java card 3 is commercially available, those cards may constitute an alternative to .NET cards.

Chapter 4

Distributed timestamping

This is the second of two chapters in the area of time synchronization and timestamping. In the previous chapter we introduced our interval-based timestamping approach. In this chapter we improve on the original protocol, by introducing a distributed timestamping protocol that timestamps data with multiple smart cards, instead of only a single smart card. This allows us to better mitigate physical attacks against smart cards.

In our application scenario distributed timestamping is an alternative to our original interval-based timestamping approach. The applicability depends on the relation between the cost to successfully launch a physical attack against a single smart card and the maximum gain possible with such an attack. In cases where the gain is higher than the cost distributed timestamping is more appropriate than interval-based timestamping.

Distributed timestamping influences our original approach of trading dependability and security, by increasing security at the cost dependability and thereby shifting the trade-off back in the opposite direction. Security increases, as it is not sufficient for an adversary to manipulate only a single smart card. However, this comes at the cost of dependability, as distributed timestamping cannot be used during outages of the client's Internet connection. Depending on the required security vs. dependability trade-off it is possible to adaptively adjust the parameters used by our distributed timestamping protocol. For example, increasing the amount of timestamps required for each bid increases the protocol's security, but also decreases dependability.

First, Section 4.1 gives an introduction to distributed timestamping. Then, Section 4.2 discusses our timestamping protocol with a focus on our overlay routing approach and efficient connection establishment. Based on this,

Section 4.3 continues with the evaluation using our prototype implementation and a protocol simulation using the PeerSim [MJ09a] network simulator. Finally, Section 4.4 discusses related work and Section 4.5 concludes the chapter.

4.1 Introduction

Timestamping protocols allow to certify that a particular document existed at a particular point in time. In practice, central timestamping servers operated by trusted third parties are typically used. While such central timestamp servers work fine in application scenarios where availability is only secondary, they exhibit a lower dependability than distributed timestamping approaches as server or network outages or overload can lead to a loss of service or to inaccurate timestamps.

Security issues in today's distributed protocols are one of the reasons why mostly centralized protocols are used, as in the case of distributed protocols each participant needs to trust that a sufficient percentage of other participants will cooperate. In addition, communication between nodes is a problem. Usually, distributed timestamping protocols are built on the assumption that each node can directly request a timestamp from every other node. However, NAT (Network Address Translation) traversal is often required for communication between nodes on the Internet, leading to increased connection setup times, and thus decreased timestamp accuracy. In addition, some of the nodes might not be active due to node churn. In a traditional distributed timestamping protocol such inactive nodes will only be detected *after* trying to send a request to these nodes, which can lead to inaccurate timestamps.

In this chapter we present our distributed timestamping protocol. Untrusted terminals are used for network communication and as interface to the user, while smart cards are responsible for timestamping, cryptographic aspects, and application level routing decisions. Thus, even an attacker with full control over a user's terminal has only limited options to attack our protocol. In addition, we provide an efficient overlay routing protocol, which trades simplicity in connection setup and maintenance for a lower latency in timestamp transmission. While pre-establishment of outgoing connections leads to higher bandwidth and computational requirements, it allows to efficiently route requests along pre-existing connections, without connection establishment or node lookup delays. However, this comes at the cost of a higher impact of hop-to-hop latencies, as messages now have to be routed along several hops.

4.2 Timestamping protocol

This section discusses our distributed timestamping protocol that enables accurate and secure distributed timestamping in real world application scenarios. In comparison to the state-of-the-art, our protocol allows for minimum latency when deployed on the public Internet. In addition, we increase the security by restricting which nodes are allowed to timestamp particular messages.

In comparison to traditional "k among n" based protocols [BLGB06]—where k timestamps are obtained from a set of n nodes—our protocol benefits from the following characteristics:

- **Deterministic calculations without global state**

  We deterministically assign nodes responsible to timestamp a particular document allowing to subsequently verify if a given document has been timestamped by the correct nodes. Only an agreement on the *approximate* size of the network is required, instead of a global agreement on the set n.

- **Small latencies between hosts**

  Our protocol takes real-world network conditions into account and is optimized for the fact that connection establishment can be expensive due to NAT traversal. As we pre-establish connections, connection setup times do not influence the accuracy of timestamping. Furthermore, pre-established connections allow to use keep-alive messages as simple failure detector to detect unavailable nodes.

- **Anonymity**

  While full anonymity is not a goal of our protocol, we strive for semi-anonymity to make it more *difficult* to assert the originator of a timestamping request. This prevents individual participants from, e.g., deliberately delaying requests of certain other participants.

- **Smart card based security**

  Existing distributed timestamping protocols typically assume semi-trusted nodes for the application of each other's timestamps, which is often not a reasonable assumption. Therefore, we decrease trust requirements by combining the user's terminal with a smart card responsible for security-critical functionality.

To provide these benefits, we take the following trade-offs in comparison with a traditional "k among n" approach into account:

- **Increased hop count**

  In comparison to "k among n" approaches, which do not route messages along multiple hops, we increase the overall length of the routing path and thereby the influence of latency on the timestamp accuracy. However, we keep hop-to-hop latencies low by pre-establishing connections. Furthermore, nodes can detect slow neighbors with our keep-alive mechanism, allowing to replace them with faster nodes.

- **Higher overhead**

  There is a higher total overhead due to the pre-established connections and the use of a DHT (Distributed Hash Table). However, this overhead does not affect the accuracy of timestamping requests and is a deliberate trade-off.

In the following subsections we first give a protocol overview, followed by an examination of our Node IDs and how values derived from these IDs are calculated. We then continue by discussing our network structure, message routing, and connection establishment techniques.

4.2.1 Protocol overview

In our protocol each node is represented by the user's computer under full control of the user, as well as the trusted smart card emitted by a trusted third party. While the main protocol tasks are executed by the user's terminal, the smart card is responsible for the timestamping itself. In addition, the smart card is also able to restrict routing decisions by the terminal. Apart from users, nodes can also be operated by trusted third parties to enhance the security of the protocol by providing more trustworthy timestamps.

Our protocol is implemented as overlay network, with timestamp routing performed on this overlay network. The network is partitioned into sets of address ranges, where each node belongs to exactly one particular address range. To mitigate node churn, address ranges can be reassigned (see Section 4.2.2) when the total number of nodes changes, so that the number of nodes within an address range is relatively constant. Each outgoing connection of a node has a unique index number and is established to one node within a deterministically identified address range. The respective address range to which a

connection is established is determined by factors such as the index number of the outgoing connection, the local node's own address range, and the current time. Thus, a node is restricted in the choice of outgoing connections. Open outgoing connections are regularly recycled to increase diversity in routing paths.

When a timestamp request arrives, the hash of the request determines the routing path, which corresponds to the index numbers at the respective nodes. Thus, timestamp requests can be efficiently routed along the already pre-established connections, without needing to establish new connections. Due to the deterministic routing decisions, external nodes can verify if timestamp requests have been routed along the correct path.

4.2.2 Node IDs

A fundamental security feature of our protocol is to restrict the nodes that are allowed to apply a timestamp for a given document. Typically, such restrictions are implemented with "k among n" schemes [HS91], where k nodes are required that can be chosen from a set of n nodes. In our case we determine the set n from factors such as the hash of the local node and the current time. We cannot directly use IP (Internet Protocol) addresses in this set, as these addresses are sparse and unevenly assigned, which makes it difficult to define a function that returns an address range with a predefined number of active nodes. Definition of such a function would only be possible with the knowledge of all active node's IDs.

Instead, we use an overlay network and perform the operations on overlay node IDs, which are assigned in a circular address space. IDs are derived by hashing the user's public key and are thus randomly distributed. As the public key pair of a node is generated directly on the smart card while the card is still under physical control of the trusted third party, the user cannot use brute-force attacks to force a node ID in a chosen ID range.

Furthermore, each node ID is member of exactly one address range, with the whole address space partitioned into r different ranges. While for each particular point in time there is a deterministic mapping between node IDs and address ranges, r can change over time, thereby affecting the mapping of node IDs to address ranges. In practice, r will be adapted to the total number of nodes, leading to a relatively stable amount of active nodes within each particular address range. Depending on the concrete application scenario there are different approaches to estimate the total amount of nodes. For example, if the Kademlia DHT [MM02] is used, trusted nodes can estimate

the amount of total nodes by looking at the Kademlia bucket utilization and then cryptographically sign and broadcast this information via the overlay.

4.2.3 Derived values

In our protocol we use different *derived* values to determine values such as address ranges. These derived values are used in place of *random* values (e.g., to determine the routing path). Because of the deterministic calculation, dishonest nodes cannot choose values at will, as external observers can verify if the correct rules have been used to obtain the values. In this section we specify how we map these input values to their respective output values.

Derived values are the result of a hash function with the source element from which a value is derived combined with an index used as the input. The index serves as distinguishing element when the same input element is used to produce multiple output elements. For example, to derive an address range for the fifth outgoing connection from the node's own address range we calculate $new\_range = hash(local\_range + index)$, where new_range identifies the resulting address range, $local\_range$ identifies the node's own address range, and $index$ is set to 5 to represent the fifth outgoing connection.

In related work sometimes pseudo-random number generators are used for a similar purpose in "k among n" schemes [HS91]. However, using a hash function allows for non-sequential access to the individual values, while a pseudo-random number generators would require the calculation of all preceding values. For example, if the smart card is used to verify the value of a particular outgoing connection. Furthermore, on smart cards the respective optimized native implementations can be used for hash functions, while the use of pseudo-random number generators would require access to the generator's seed which is often not possible.

4.2.4 Network structure

Our network is built as an overlay network between nodes. At each point in time each node has exactly c_o outgoing and approximately c_i incoming connections. When a node sends a timestamp request to an outgoing connection, the request arrives at the incoming connection of another node. While communication channels are bidirectional, requests only travel along outgoing connections, while replies travel in the reverse directions.

When a new connection is established, this step is split into two tasks: First, we use a mapping function that takes (i) the particular index of the outgoing connection, (ii) the node's own address range, and (iii) the time as input. The output of of this function is the address range to which the connection will be established. Afterwards, the terminal looks up one of the nodes within this address range and establishes an outgoing connection. For even better handling of node churn, redundant connections into the same address range can be established for immediate failover. This approach is explained in detail in Section 4.2.6.

Each of the outgoing connections is assigned an index from 0 to $c_o - 1$. The routing path is calculated by the source node and included within the message. For example, a source node may specify that a message should travel along the edges 1–2. The source node then relays the message to its outgoing connection with the index 1, where the node receiving the message continues relaying the message to it's own outgoing connection with the index 2.

The exact lookup mechanism used by the protocol to find nodes within particular address ranges is not fixed and can vary across implementations. For example, if a DHT such as Kademlia is used, a *range query* can be used to find suitable candidates, while in the case of JXTA [JXT] advertisements stored in a SRDI (Shared Resource Distributed Index) would be used.

4.2.5 Message routing

This section describes how a message for a particular timestamp is routed along the nodes. First, the node requesting a timestamp uses the hash of the document to calculate the routing path of the message. For the routing path each hop is iteratively calculated with the formula $hop_{next} = hash(document\_hash + hop\_cnt)$, where hop_cnt indicates the index of the particular hop. The routing path is then embedded within the message that is to be timestamped. Before sending a message, the originating node removes the first element from the routing path, and forwards the message on the outgoing connection that is identified by the element. Each node receiving the message in sequence removes the first element from the routing path and forwards the message to the respective outgoing connection. Once the routing path in the message is empty the message is not forwarded any further. An example of this routing strategy is given in Figure 4.1 where the root node calculates the routing path "1-2".

To mitigate message loops where a message is relayed back to a node that has already received this message in a previous step, the first node originating

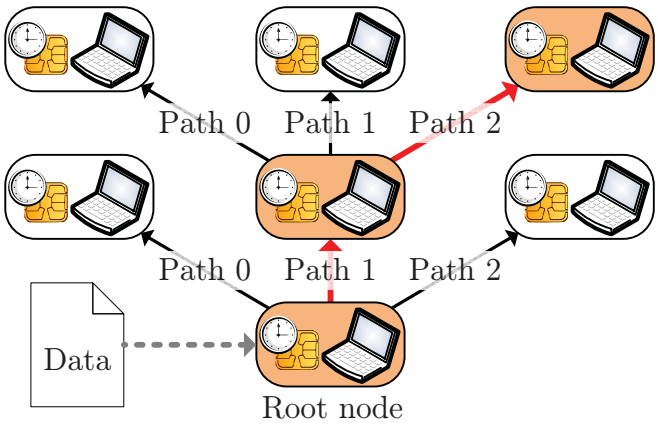


Figure 4.1: Path selection: Path 1-2

the message validates that each address range is only traversed once. To do so, the node simulates the message path, as it can locally calculate to which address range an outgoing connection of another node will connect. If a potential loop is detected, because the message is relayed back to an address range from which it has previously been forwarded, the *hop_cnt* used in the formula above is incremented, thus shifting the *hop_cnt* of the affected element and all remaining elements by one. This algorithm is deterministic and only depends on the partitioning of the system into the different address ranges. Therefore, it allows to externally verify that the correct routing path was used. This verification depends on the cryptographically signed address range partitioning parameter r (Section 4.2.2) that is embedded within the timestamp.

When a node sends or relays a message, the message itself is generated and signed by the smart card. This step is required to prevent malicious nodes from *injecting* bogus messages into the network. Therefore, the message also needs to contain the concrete node ID of the next node, as otherwise the user's terminal would be able to inject a single timestamping message to multiple nodes within the same address range. The signatures applied by smart cards are only used on a hop-by-hop basis. Thus, removal of an element from the routing list does not affect signatures applied by a preceding hop.

When a node receives a message from a neighboring node it first verifies if the message has indeed been signed by its neighbor and if the existing timestamp(s) in the message are still current. Additionally, it verifies if the destination field of the message matches its own local node ID. Before forwarding the message, the node locally stores a timestamp for this message

and the node ID from the node from which the message was received, but does not include this timestamp in the relayed message. When the last node on the routing list receives the message it includes a timestamp and relays the message back to the node from which it has been received. Each node then includes its previously stored timestamp and continues relaying the message in the reverse direction. When the message arrives at the initial originator of the request, the message contains the timestamps assigned by all the nodes along the routing path.

4.2.6 Connection establishment

For regular connection establishment we use an approach that continuously reestablishes connections into new node ranges to provide diversity in the possible routing paths. With each step only one single connection is reestablished, thereby minimizing the impact on the network setup. In addition, we adapt our routing approach to prevent routing paths over outgoing connections that will be renewed soon, as this would prevent the sender from uniquely determining the routing path, which is required for loop mitigation.

For example, consider that each node has 10 outgoing connections and that within a 10 minute interval each outgoing connection should reconnect to a new address range. When updating connections continuously, we can reconnect a single connection once per minute. To avoid any routing problems due to reconnecting nodes, a node can establish a connection to the new address range in advance, and then just switch connections when the interval of the current connection ends. If the node is still waiting for a reply message that was initially forwarded over the old connection, it can additionally leave the old connection open until the reply message has been received.

While the renewal of a connection does not directly affect messages in transit, this renewal can lead to routing loops, as the original mitigation strategy presented in Section 4.2.5 only considers the network configuration at the time of message creation. To mitigate such loops, we adapt the routing mechanism not to use connections that will be renewed *soon*, e.g., to never use the next connection that will be renewed. In such a case a similar strategy as for loop mitigation is used: hop_cnt is incremented, causing the algorithm to calculate a different route. Of course, such an approach requires that all nodes have a synchronized time. This is not a problem in our case, as such a synchronized time is already required for timestamping purposes.

To account for node churn, e.g., active nodes that change their status to inactive, we use regular keep-alives to detect stale connections. In this case a

node will re-establish the connection to another node within the same address range. Such a connection establishment cannot lead to a routing loop, as the address range of the old connection and the new connection is the same. While node churn leads to overhead due to higher reconnect rates, it does not affect latency, as messages are routed only along pre-existing connections.

4.3 Evaluation

The evaluation of our protocol was conducted in two distinct steps. First, we used a prototype implementation to show the feasibility of our approach and for performance tests with low amounts of nodes. This implementation is based on JXTA [JXT]. The *Netem* Linux kernel module is used to simulate network latency. Second, we used the PeerSim [MJ09a] network simulator to examine the effects of latency and churn. Significant parts of the prototype implementation's code have been reused in the simulation, ensuring that the simulation's behavior matches the prototype. We also validated if the results of the simulation match the results obtained in the prototype implementation.

Figure 4.2 shows a comparison between the average latency in our overlay routing approach and an alternative approach that requires connection establishment for forwarding. To model node-churn, we assume a node-failure rate of 5%. If a message is transmitted to a *failed* node, it cannot be processed, but is re-transmitted to another node.

In the results we observe that the latency without an overlay network is roughly twice as high as the latency in our overlay approach. While in time synchronization protocols latency can be mitigated by assuming synchronous network delays, this is not possible in timestamping. Therefore, the latency directly affects the timestamps assigned to individual documents. While in some application scenarios timeliness of timestamps is only secondary—e.g., when only an approximate date is required—in other application scenarios such as our deadline-based online auctions timeliness is a fundamental property.

In addition, our simulation has shown that the results also depend on the assumed application scenario: In application scenarios without node churn where UDP communication without NAT traversal can be used between nodes, the differences between a traditional approach and our approach are smaller, while in application scenarios with considerable node churn the differences are higher. For example, if we decrease the node failure rate from 5% to 0% in the scenario *without* overlay network, this decreases the average delay at the fifth hop from 355 ms to 338 ms, which is only a marginal improvement.

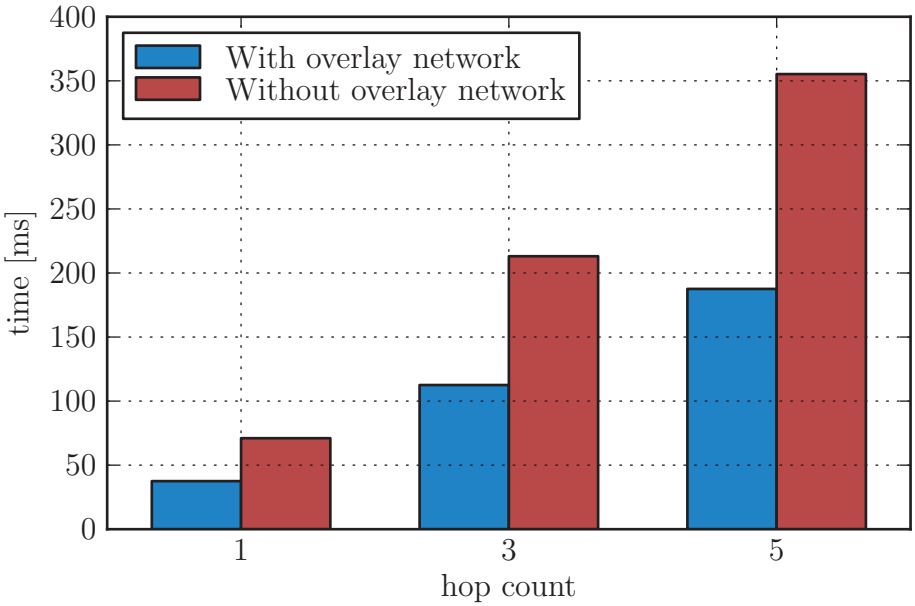


Figure 4.2: Hop latency

On the other hand, if we increase the node failure rate to 30% this increases the average delay to 482 ms. However, even in the first example with lower differences our protocol allows for more efficient routing without requiring nodes to know all other nodes, while a traditional protocol would require nodes to store information about other nodes, if nodes are to be selected according to the document's hash.

To examine the effects of node churn we simulated our protocol with different median session times and tracked the number of failed request. A request fails, if a node sending a request does not receive the corresponding response, because an intermediate node cannot forward the request as no outgoing connections to the respective node range are available. For our tests we assumed a hop count of 5 hops as well as 3 redundant connections into each node range. The results in Figure 4.3 show that request failure rates are an issue for median session times below approximately 40 minutes. While failure rates are high for very low session times, they decrease to less than 5% for median session times of 20 minutes or more and to less than 1% for median session times of 40 minutes or more.

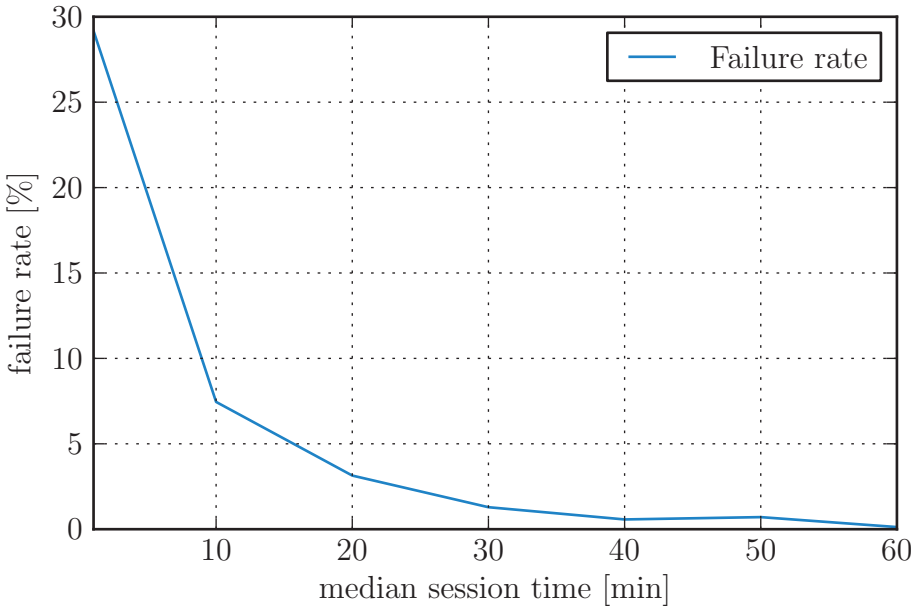


Figure 4.3: Median session time vs. failure rate

4.4 Related work

Distributed timestamping protocols are discussed in [HS91, MAQ99, BLGB06, GKŁ06, Tul06, LBG08, NM08], but only Nishikawa and Matsuoka [NM08] seem to actually have implemented their protocol and also consider lower level issues such as techniques for timestamp transmission. As the protocol requires individual TCP connections, it leads to higher latency than our overlay approach. [BLGB06] presents a "k among n" timestamping scheme with redundancy to mitigate failed servers. Unlike our approach it requires synchronization between servers and does not mitigate delays during connection establishment. [Tul06] suggests a timestamping scheme with better scalability than classical tree-based linking scheme. In their work client latency depends mainly on the transmission delay which can be reduced with our approach. While [GKŁ06] uses onion routing over an overlay network for timestamping, it differs from our approach as the path of the onion route is not dependent on the input document and hence does not allow to specify and verify whether correct timestamping nodes were chosen.

4.5 Conclusions

In this chapter we presented our protocol for secure distributed timestamping in our online auction scenario. The initial motivation for the protocol was to increase the security of interval-based timestamping as discussed in Chapter 3, in cases where the security provided by smart cards is not sufficient.

The main goal for the protocol is to provide a secure low-latency mechanism for document-dependent routing and to decrease information required at individual nodes to facilitate implementation on smart cards. As a solution approach we decouple the setup of the overlay network from the underlying routing mechanism. This allows for variable routing paths, while at the same time allowing the system to route all messages along pre-existing connections, which leads to a higher overhead for connection setup and connection maintenance. Our evaluation has shown that routing messages along existing connections on an overlay network provides for considerably better performance than state-of-the-art protocols.

Chapter 5

Transaction authentication

This is the first of two chapters in the area of secure communication and transaction authentication. The goal of those two chapters is to improve the security from an honest bidder's point of view, e.g., by ensuring that malware or other malicious bidders are not able to manipulate placed bids. In this chapter we present our QR-TAN (Quick Response - Transaction Authentication Number) transaction authentication technique that allows bidders to use a secure device to verify and confirm the content of their transactions.

In our application scenario QR-TAN is an essential component, as successful attacks against an auction reduce the reputation of the auctioneer, even in cases where attacks are enabled by weak security at a bidder's terminal. In addition, QR-TAN increases the non-repudiation properties of bids, as the increased security makes it harder for a bidder to plausibly claim that a given bid has been placed or been manipulated by malware.

In Section 5.1 we give an introduction to our QR-TAN protocol, followed by Section 5.2 where we provide a problem definition. We then discuss related work in Section 5.3 and contribute with our QR-TAN protocol in Section 5.4. Afterwards, we analyze specific attacks against today's smart phones when used as secure device for QR-TANs in Section 5.5. We discuss related work in Section 5.6 and conclude the chapter in Section 5.7.

5.1 Introduction

In traditional signature mechanisms, the user who applies a signature has full control over the signature process. In contrast, in the case of electronic signatures the user depends on a client that often cannot be trusted. Even if a secure smart card is used, the user is often not able to assert that the information displayed on the screen is actually equal to the information signed by the smart card.

This problem is present in all types of electronic transactions that require some type of signature by the user. Examples include online banking and electronic signatures for contracts. The main problem is that information on the client can be arbitrarily modified by malicious software. The article *Secure Internet Banking Authentication* [HKW06] by Thorsten Kramp and Thomas Weigold provides a taxonomy of techniques classified by resilience against offline and online attacks. The only evaluated technique that is robust against content-manipulation attacks is *transaction-signing*. This method requires the user to execute critical operations on a trusted reader device.

In this chapter we propose an authentication technique called *QR-TAN*. QR-TANs use a *transaction-signing* method that has been adapted to fit the capabilities of commonly used Web-based applications. QR-TANs are based on two-dimensional QR (Quick Response) barcodes. Similar to other approaches [BF99, MPR05, MvO07, CGK+02, OBDS04], QR-TANs authenticate transactions by using a trusted device. This device can be a mobile phone with a display and a camera with a modest resolution. QR-TANs use QR codes for the transmission of information.

If smart card technology is combined with QR-TANs, transactions can be conducted completely offline without any network connection. QR-TANs do not require any special hardware support on the terminal and could thus easily be used by any kind of Web application. QR-TANs improve on the properties of mobile TANs by not requiring any networking capabilities on the trusted device and by using secure encryption techniques to provide security if an attacker gains access to the trusted device.

5.2 Problem definition

The main problem in today's electronic transactions is that the user cannot fully trust the terminal. An attacker might be able to control the terminal and to manipulate any transaction. Therefore, the user cannot assert that the

authentication is applied to the same data as the data displayed on the screen. In *Secure Input for Web Applications* [SKK07] the three phases of an attack against a user's computer are described. In the first phase, executable malware is installed on the computer. In the second phase, the malware monitors the user's interaction with Web applications. If the malware detects a security critical operation, it modifies or captures the transmitted information in the final phase.

For most electronic transactions, it is in the interest of both parties that the transactions cannot be forged. Furthermore, it should not be possible for a party to repudiate a transaction. The term *non-repudiation* means that a party must not be able to dispute such a transaction after it has been completed. In order to guarantee *non-repudiation* in our application scenario, we must not only ensure the security of the auctioneer's systems, but also the security on the client side. Therefore, malicious software must not be able to create transactions that have not been approved by the signing party.

We assume that an attacker has full control over the computer and may read and modify any message transferred between the user and a trusted server.

5.3 State of the art

In the past, different approaches have been suggested and used to address the problem of the untrusted client. Many of them have been developed in the context of online banking applications. This section discusses the state of the art of techniques that are currently used in commercial settings.

TAN codes

TAN codes have traditionally been used by banks to prevent attackers from using a captured password to authenticate transactions. Each user receives a private list of TAN codes and each code may only be used once. An attacker cannot use a captured TAN code if it has already been used before. TAN codes are not able to assure the content of a transaction. A malicious terminal might accept transaction data and a TAN code from a user, but then relay different transaction data to a server. Furthermore, phishing attacks may cause users to provide their TAN codes to an attacker.

Mobile TANs

Mobile TANs are an alternative to traditional TANs. When a user needs to authenticate a transaction, the bank sends a summary of the transaction in combination with a TAN code via SMS (Short Message Service). The user then verifies that the summary matches the intended transaction and enters the TAN code into their computer. Hence, the advantages of mobile TANs in comparison to traditional TANs are that users do not need a TAN list and that users can use their mobile phones as secure device to approve and validate transactions.

However, mobile TANs also exhibit several disadvantages. First, they result in additional costs for the transmission of messages. Furthermore, there are also problems with the encryption used in mobile networks. The GSM (Global System for Mobile Communications) protocol uses the A5/1 and A5/2 ciphers that suffer from well-known vulnerabilities. A5/2 can already be cracked in real-time with a cipher-text only attack [BBK03]. According to a presentation held by the security researchers David Hulton and Steve Muller at the Black Hat 2008 security conference in Washington, D.C., A5/1 can be decoded in about thirty seconds with equipment that consists of 16 128 GB (Gigabyte) flash hard drives and 32 FPGAs (Field Programmable Gate Arrays).

Security tokens

Security tokens are small electronic devices that can be used for two-factor authentication. Typically, they are either directly connected to a computer or a time-dependent number displayed by the device has to be entered into the computer. Security tokens suffer from the same vulnerabilities as TAN codes: The user has no possibility to verify that the data displayed by the computer is actually equal to the data validated with the security token.

Smart cards

Smart cards are credit-card sized devices able to perform cryptographic operations. They are typically used to apply digital signatures or to provide two-factor authentication to a system. A smart card usually does not provide a screen or a keyboard. Therefore, a terminal is required for communication with the user. Unfortunately, smart cards do not generally prevent attacks performed on untrusted terminals, because man-in-the-middle attacks on the

terminal may modify any information transferred between the smart card and the user [SS99].

5.4 QR-TAN

In order to address the shortcomings of existing solutions, we propose QR-TANs as new transaction authentication technique based on two-dimensional barcodes. QR-TANs allow a user to validate and approve a transaction using an untrusted terminal connected over an untrusted network. We do not place any requirements on the security that must be provided by the terminal. In fact, even if an attacker would be able to fully control the terminal, this would not decrease the security provided by QR-TAN.

This section first gives a short introduction to QR codes. Afterwards, the QR-TAN protocol used for transaction authentication is described, followed by a discussion on the design decisions and implementation details.

5.4.1 QR codes

QR codes have been invented by Denso Wave Incorporated[1] in 1994. The two-dimensional structure of QR codes allows for codes with a lower resolution in any single dimension than a comparable one-dimensional code. Therefore, they can be better recognized by cameras, as the resolution of these cameras is typically the same in both directions. In contrast, one-dimensional barcodes require higher resolution cameras, as more information is stored in a single dimension.

There are different QR code *versions* that mainly differ by the amount of data that can be stored and by the size of the two-dimensional barcode. For example, version 1 contains 21x21 modules and can encode up to 25 alphanumerical characters. Version 2 has a size of 25x25 modules and can encode up to 47 alphanumerical characters. The largest amount of data can be stored in QR codes of version 40 with a size of 177x177 modules, they can store up to 4,296 alphanumeric characters (or 2,953 binary 8-bit characters). The practical limit on the size of a QR code is the camera used to capture the code. The main factors are the resolution of the camera and whether the camera is able to focus on an object. Figure 5.1 shows an example of a QR code over the "data+nonce" arrow that contains the string "*QR-TAN!*".

[1] `http://www.denso-wave.com/qrcode/`

5.4.2 QR-TAN protocol

The QR-TAN approach uses a challenge-response mechanism to validate individual transactions. The challenge is transmitted to the phone by the use of two-dimensional barcodes, the response is a few-letter code typed into the computer by the user. A malicious man-in-the-middle is able to learn information about individual transactions, but is neither able to create new transactions nor able to modify existing transactions.

Figure 5.1 shows the messages that are transmitted during the authentication process. In order to sign a message, the following steps are executed:

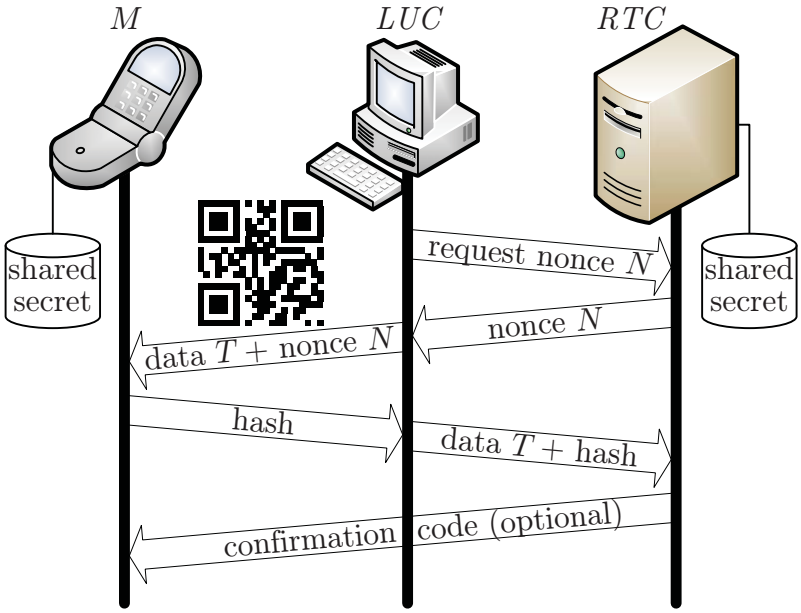


Figure 5.1: QR-TAN authentication

1. The user U, who intends to perform a transaction, generates the transaction data T on the local untrusted computer LUC. It is assumed that an attacker is able to read and modify any data stored or relayed by the LUC.

2. The LUC requests a nonce N from the remote trusted computer RTC, e.g., a bank's server. The nonce is required to prevent replay attacks and to prove the freshness of transactions. The RTC is the trusted device in charge of the transaction. In the case of offline transactions a smart card could also perform the tasks of the RTC.

3. The LUC concatenates T and N, encrypts them with the public key of the user's mobile phone M and displays the result as a QR barcode.

4. U uses the mobile phone M to extract T and N and to read T. M acts as a trusted device.

5. To approve T the user enters the secret password on M to decrypt the device password D. The device password is a shared secret between the device and the server. It can be initially distributed to the user via letter post as a QR code.

6. M calculates $HMAC_D(T + N + approve + cnt)$ [KBC97], converts the result to an alphanumeric format and displays the first X characters. The *approve* value serves as an indicator that the user wants to accept the transaction. If the user prefers to explicitly reject a transaction, they can calculate the hash value of $HMAC_D(T + N + reject + cnt)$. The *cnt* field is usually set to zero. However, if the resulting shortened hash values for *approve* and *reject* match, *cnt* is increased until the two hash values differ.

7. U reads the first X characters and inputs them on LUC.

8. LUC transmits T and the hash to the RTC.

9. RTC does the calculation for the two possible hashes in step 6 itself and checks if the received hash matches one of the two hashes. Furthermore, the RTC computes the confirmation hash $HMAC_D(T + N + check)$ and transmits it back to the LUC.

10. M computes the confirmation hash using the same formula as the RTC. If the hashes displayed by M and LUC match, the user knows that the transaction has been confirmed by the RTC.

5.4.3 Design decisions

One of the main design decisions was that manual work required by the user and computational requirements on the trusted device should be kept to a minimum. Unlike other approaches that require the user to perform manual calculations or approaches that require a separate channel, e.g., via Bluetooth, QR-TAN only requires the user to validate the transaction on their trusted device and to approve the transaction by entering a short number into their

computer. The effort required by the user to learn the QR-TAN mechanism is therefore comparable to other TAN based approaches.

In order to design a technique that is also applicable to commercial applications, we placed several requirements on the resulting QR-TAN protocol:

- QR-TANs must not require any manual computations or other complex tasks by the user.
- QR-TANs must provide the same (or a better) security than other existing established approaches.
- It should be possible to use QR-TANs offline by embedding the server side tasks within a smart card (only standard algorithms like AES (Advanced Encryption Standard)/SHA-1 (Secure Hash Algorithm 1) should be used).
- Implementation and usage should be possible with only modest expenses. Operation should be cheaper than the commonly used mTAN (Mobile Transaction Authentication Number) technique.

When comparing the transport mechanism, there is an important asymmetry. While the path to the mobile phone could easily convey several tens of bytes to several hundreds of bytes, the path from the phone to the computer can only transport a few bytes. The reason is that the user manually needs to enter the information displayed on the phone onto the keyboard. Therefore, this restricts how the QR-TAN technique is designed and which cryptographic techniques can be used. For example, if the output is encrypted with public key cryptography or a block cipher like AES, there is a minimum length for the cipher text. In most situations, however, expecting that a user manually types information of this length into the computer is not an acceptable solution.

5.4.4 Discussion

This section discusses details affecting the security of QR-TANs as well as selected implementation details.

Encryption

Most of the data are not signed or encrypted as these security measures would not provide any additional security. The nonce can only be used by M. An

attacker who knows the nonce cannot generate valid hashes or replay any transactions. If an attacker manipulates the nonce, this would cause M to generate an invalid hash. The same is true for the transmitted hashes as knowledge of these hashes does not provide any viable information to an attacker.

A nonce is required to prove the freshness of the data and to prevent replay attacks. If it is not possible to obtain a nonce from the RTC, the LUC could instead include a timestamp in the data. The user then verifies the timestamp and only signs the data if the timestamp is correct. When the data are transmitted to the RTC, the LUC also includes the timestamp that is part of the hash. The RTC can then check if the timestamp agrees with the hash and if the timestamp is near the current time. By rejecting identical transaction data with a timestamp that is identical to another timestamp, during the validity period of both timestamps, the RTC can prevent replay attacks.

The purpose of the public key algorithm used to encrypt messages to the mobile phone is to prevent some types of attacks. For example, an attacker who is physically located near the user, but who is not able to decipher the individual characters on the screen might still be able to decode the displayed QR barcode.

For the symmetric encryption of the data, any encryption scheme could be used. For the asymmetric encryption, a public key algorithm that yields a small ciphertext would be advantageous. Therefore, if possible ECC (Elliptic Curve Cryptography) should be used instead of RSA. When compared to RSA, the ciphertexts generated by ECC are smaller for a given plaintext and for a given level of security.

The password used to decrypt the shared secret for the message signatures must not be related to the decryption of the QR codes read by the mobile device, e.g., by securing the private key with this password. Otherwise, if an attacker would be able to obtain the device, to extract the private key, and obtain a picture of a QR code destined for the device, the attacker could brute-force attack the user's password and verify the result by checking if the decrypted QR code contains reasonable data. This password could then also be used by the attacker to sign transactions. If the user's password is only used for signatures, the attacker has no prior indication if the password is correct. Therefore, the RTC could block the account if there are more than a given number of failed attempts.

Resilience against attacks

For the generation of the hash displayed on the mobile phone and transmitted by the user to the *LUC*, the binary data needs to be recoded to a format that can easily be handled by the user. As a trivial solution, this data can be converted to a format that contains only case-sensitive alphanumerical characters. This would yield 62 different possibilities per position. If two of these possibilities are removed in order to prevent user's from mistaking *l* with *1* (one) or *O* with *0* (zero), this leaves 60 possibilities per position. With a hash length of four characters the total number of combinations is 12 960 000. The chance of guessing the correct code is therefore roughly comparable to the chance of winning the jackpot in a 6 from 49 lottery. A hash length of six characters would already yield 46 656 000 000 different combinations. As the number of accepted trials by the *RTC* can be set to a low value, this effectively prevents brute force attacks that could be used to guess the correct hash.

Hash options

The decision if a user wants to *approve* or *reject* a transaction is encoded in the hash as this prevents the *LUC* from learning the information. Depending on the application, additional values may be used. If it is allowable that the *LUC* is able to learn the choice of the user, a user could also omit the input of any hash value. Consequently, this would lead to a timeout for the nonce at the server.

The confirmation hash generated with the *check* option allows the user to verify that a transaction has been received by the *RTC*. It can be displayed as a shortened hash or by using hash visualization techniques [PS99]. The confirmation hash helps the user to discover that the *LUC* does not forward transactions accordingly. However, if a user does not receive any confirmation hash by the *LUC* they cannot detect if the message containing the transaction, or the message containing the confirmation was lost. This is a more general problem related to the Two Army Problem [AEH75] that cannot be fully solved.

5.5 Attacks and security

In this section we analyze potential attacks against smart phones used as secure device for QR-TAN authentication. First, we discuss smart phone security issues in Section 5.5.1, followed by Section 5.5.2 analyzing the consequences of these security issues for QR-TAN.

5.5.1 Smart phones

While modern smart phone operating systems typically include security models that are more advanced than the security models of standard desktop operating systems, these security models can only protect against malicious software if there are no exploitable bugs in the operating system itself. However, in the past iOS and Android contained security problems that allowed arbitrary applications to gain root privileges. Moreover, some of those problems have been unpatched for several weeks even though they were publicly known.

Examples of recent security problems are the combination of two iOS exploits, where the first exploit [CVE10a] allows to execute arbitrary code on the system and the second exploit [CVE10b] allows code running on the system to gain system privileges. As the first problem concerns the PDF (Portable Document Format) rendering engine of the Web browser, it cannot only be abused by local applications, but also by arbitrary web sites. On Android a similar exploit was possible for an installed application by installing itself as callback function for the hotplug functionality. Thus, upon enabling or disabling any hotplug device this application is executed with system privileges. However, unlike in the case of iOS this problem could could not be exploited remotely.

In addition to the security issues stated above, social engineering is an issue that needs to be considered as well. For example, variants of the ZeuS malware first infect a user's PC and then prompt the user to install a software update on their smart phone [Bun10]. However, this software update is not genuine, but does itself contain malicious code. Thus, an adversary is able to control both devices used for two-factor authentication.

5.5.2 QR-TAN

In *Two-Factor Authentication: Too Little, Too Late* [Sch05] Bruce Schneier argues that even modern two-factor authentication techniques will not solve today's phishing problems as they do not solve man-in-the-middle attacks and

trojan attacks. This reasoning is valid if the user has no possibility to assert that the content of a transaction is correct. However, in the case of QR-TANs a user directly validates each transaction on a secure device. Therefore, the user can check if the transaction displayed on the secure device matches the transaction that has been input in the computer.

Attackers are only able to calculate the hash code, if they are able to obtain the key shared between the user's phone and the *RTC*. However, the key is protected by the user's personal password and optionally also stored within a secure element (e.g., on a smart card chip).

To successfully issue a malicious transaction, an attacker would need to (i) know the password required to login to the bank's website, (ii) steal the mobile phone of the user and bypass access controls (e.g., the password of a screensaver), and (iii) know the personal password of the user required to approve transactions on the phone.

If the mobile phone is considered insecure, for example because an attacker might be able to execute software on the phone as discussed in Section 5.5.1, the attacker would still need to gain control over the phone *and* over the terminal. If the cryptographic algorithm as well as the required user credentials are stored within the phone's smart card, access to the phone is required at the time of the transaction.

For high security applications it might therefore make sense to use a device similar to security tokens for the authentication of transactions. This device could feature a simple camera used to scan QR codes and a display to show TAN codes. The device does not need any connection to the Internet or other types of networks. Therefore, the attacker would require physical access to the device in order to install malicious software.

5.6 Related work

This section describes related work to our QR-TAN transaction authentication technique. A number of other papers propose methods that can be used to implement secure transaction authentication on untrusted terminals. There are two different types of approaches: Some of these methods use external devices that are used to validate transactions. They therefore shift the security burden from an untrusted device to an external trusted device. Other approaches require the user to manually validate that a transaction is correct.

The Axsionics Internet Passport™ [MJ09b, Mül08a] is a secure mobile device that provides mutual authentication and transaction authentication. Unlike QR-TANs, it does not use QR codes to transmit information, but a *flickering code* [MJCLR07] that can be decoded with the help of optical sensors. Therefore, no camera is required and simpler optical sensor elements are sufficient. Axsionics describes their security token as *three-factor* authentication device, as it also includes a digital fingerprint reader to identify the user. A similar device is sold by Gemalto under the name *ezio optical reader*. Unlike Axsionics' device it does not include a fingerprint reader, but allows to use smart cards for authentication. Compared to QR-TAN, the main difference to these approaches is the implementation of the optical transmission channel. In addition, while our original QR-TAN approach currently does not support *three-factor* authentication, we explicitly anticipate such authentication mechanisms by allowing to store the shared key on the phone's SIM (Subscriber Identification Module) card.

A set of related authentication systems using QR codes has been developed by Borchert and Reinhardt at the University of Tübingen [BR08]: *Sesame* allows a mobile phone to use information included within a QR code to generate a one-time password to login to a Web site. *Foto-PIN (Personal Identification Number)* works similar, but instead of generating a password it allows users to securely enter PIN codes. With this technique the users needs to enter the PIN on a shuffled on-screen keyboard, with the labels of the individual keys displayed only on the mobile phone. *Foto-TAN* works analog to Foto-PIN, but allows to digitally sign transactions. Unlike in the case of Foto-PIN it therefore also displays the transaction data on the mobile phone. Conceptually, QR-TAN correspond to Foto-TAN. However, QR-TAN can also be used to login to Web sites, similar to the Sesame approach. The main difference between QR-TAN and the discussed approaches is the communication channel from the mobile phone to the computer: While QR-TANs directly allow the user to enter a short hash code on the keyboard, the discussed approaches use a virtual on-screen keyboard, where users need to enter their code using a computer mouse.

Bottoni and Dini [BD07] use a secure device to secure transactions between a user and a merchant. While this is conceptually similar to our QR-TAN approach [4], it requires direct end-to-end communication—e.g., over SMS or UDP—between the merchant and the secure device.

Aussel et. al [AdC+09] include security-hardened monitors into applications running on untrusted platforms and use USB (Universal Serial Bus) smart cards to verify the log data provided by the monitors. While this approach

could complement our techniques for secure smart card communication (Chapter 6) it provides less security than a dedicated secure hardware device.

Lu et al. [LA04] increase security with respect to online identity theft by placing confidential information inside a smart card from where it can be transferred to a remote authenticated server. This reduces the risk of confidential information being captured by malware at a user's computer. However, it does not guarantee that data entered on the computer are not changed on the way to the server, which is the focus of our QR-TAN approach and our two smart card security extensions.

MP-Auth [MvO07] allows the user to securely login to a website and to approve transactions using a cell phone. The user's long-term secret password is entered on the trusted mobile device, while the untrusted device only gains access to a temporary secret. Unlike QR-TANs, the MP-Auth mechanism requires bidirectional data transfer between a cell phone and a terminal, such as Bluetooth or wired connections. Furthermore, software installation on the untrusted terminal is required.

Clarke et al. describe [CGK+02] how a trusted mobile device can be used to validate transactions that are conducted over an untrusted device. The main difference to QR-TANs is the way how the information is transferred from the untrusted computer to the mobile device. The paper presents two different options to read data from a screen. *Pixel mapping* establishes a mapping between the camera's pixels and the screen's pixels. Compared to QR-TANs, this approach has higher requirements on the resolution of the camera and on the computational power of the device. As a consequence, an implementation is not feasible on today's mobile phones.

The goal of *qualified mobile server signatures* [OCK10] is to provide an alternative to smart cards in citizen card applications, as smart card application are affected by issues such as the low pervasiveness of smart card readers [OCK10, Roß08]. The key approach behind mobile server signatures is similar to mTANs used in banking applications. However, instead of executing a transaction, such as a bank transfer, after a successful confirmation of the code within the SMS, the remote server signs a digital document. When compared to QR-TANs, qualified mobile server signatures have a possibly higher operating cost due to transmission of SMS. In addition, this technique is not compatible with our temporal decoupling approach, as the process of signing a bid must not be temporally decoupled due to security reasons.

5.7 Conclusions

In this chapter we presented our QR-TAN protocol for secure transaction authentication. An advantage of QR-TANs over existing authentication techniques is that they do not require any additional software installation on the terminal. While a software implementation of the QR-TAN authentication technique is required on the trusted device, this implementation does not need to be tailored to any particular service provider. Therefore, such software can be pre-installed on mobile phones or dedicated secure devices.

As we expect the following technology changes in the future, it is likely that QR-TANs can be used for a wide range of different applications.

- New cameras with higher resolutions will be available in mobile phones.
- Processing power of mobile phones will increase. Therefore, more complex barcode algorithms can be used.
- Current QR codes are black/white only. More advanced barcodes that also use different colors (e.g., Microsoft's *High Capacity Color Barcode*) are able to store more information.

In summary, QR-TANs are able to substantially increase the security of electronic transactions. User's can use their own mobile phone to approve transactions that have been created on untrusted terminals. QR-TANs use a visual communication channel and do not require any network-based communication link between the mobile trusted device and the untrusted terminal. Therefore, they do not require the installation or configuration of additional software on the untrusted terminal. This is a main advantage over other techniques that use secure devices, as these techniques often require a bidirectional link between the trusted and the untrusted device.

When compared to the established mTAN authentication technique, QR-TANs allow for less costs at the service provider while at the same time providing a higher level of security. Unlike other proposed techniques, QR-TANs only require modest communication and computation capabilities at the trusted device. Therefore, we are convinced that the usage of QR-TANs in security critical environments such as our application scenario is reasonable and that QR-TANs provide a viable alternative to today's established authentication techniques.

Chapter 6

Smart card proxy

This is the second of two chapters in the area of secure communication and transaction authentication. In this chapter we present our smart card proxy that allows to Web-enable existing smart cards. Unlike state-of-the-art software for smart card access on a PC, our approach provides a well-defined standard protocol for the proxy interface, so that a single generic proxy can be used for multiple Web applications. Therefore, client-side installation is only required once—either included in the Web browser or a single, trusted standalone application. In addition, we provide two security extensions that allow to increase the proxy's security in case of man-in-the-middle attacks, when used in combination with customized smart cards.

In our application scenario we use the smart card proxy for the communication between the Web application and the smart card, as security and convenience of existing solution approaches are unsatisfactory for our application. Due to the flexible run-time configuration of our proxy a one-time installation on a bidder's computer is sufficient, even if new functionality is added to our application or if the protocol between the Web application and the smart card changes.

First, Section 6.1 gives an introduction and a problem description. Then, Section 6.2 presents the architecture and trust model of our application. Section 6.3 introduces our generic Web mapping. Section 6.4 extends our mapping approach with TPM-based attestation, while Section 6.5 provides alternative security measures based on QR-TAN authentication. Finally, Section 6.7 concludes the chapter.

6.1 Introduction

Despite ongoing efforts to Web-enable smart cards [Lu07] there is still a *media discontinuity* when using smart cards in combination with Web applications, as smart cards typically require a *native* helper application as proxy to communicate with the Web browser. One reason is that the Web security model is fundamentally different from the smart card security model, leading to potential security issues even for simple questions such as: "Is a particular Web application allowed to access a particular smart card?".

Ongoing research to Web-enable smart cards typically either requires computational capabilities at smart cards higher than the capabilities provided by today's smart cards or requires users to install software customized to particular types of Web applications [LHP02]. In contrast, our generic mapping proxy enables access from arbitrary Web applications to arbitrary smart cards, while using access control to protect smart cards from malicious Web applications, without requiring any on-card software modifications.

However, guarding only against malicious Web applications is not sufficient, if the local computer is potentially controlled by malware. Consequently, we enhance our mapping approach to provide end-to-end security between a user and a smart card, but this enhancement requires the possibility to adapt on-card software. In particular, we allow the user to (i) either use the TPM inside their computer or, (ii) alternatively, use a trusted secure device to secure communication with the smart card.

Summarized, the contributions in this chapter are:

- A smart card Web communication protocol that provides a secure way for Web applications to interact with existing smart cards. Unlike state-of-the-art technologies, our approach allows any Web application to interact with any given smart card where communication is allowed based on our authorization and access control mechanisms.
- A first extension to our protocol that uses the Trusted Computing facilities part of recent PC hardware. This allows us to mitigate the effects of malware on the local computer, but requires modification of the on-card software.
- A second extension to our protocol that uses QR-TANs [4] instead of a TPM. Thus, the security is provided by an external security device instead of a PC. This also requires modification of the on-card software.

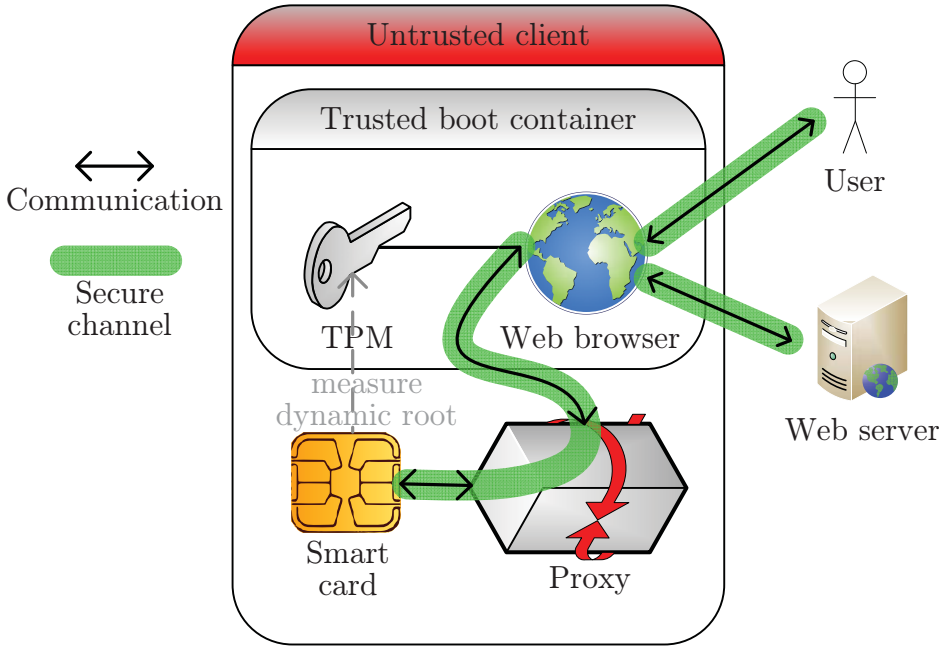
6.2 Architecture and trust model

This section presents our overall system architecture and the different security constraints. Due to the different trust requirements of the different entities, the problem we solve can be seen as a type of *multilateral security* [Ran00] problem. For example, the user and the Web server both trust the smart card, but neither does the user trust executable code provided by the Web server, nor does the Web server trust executable code provided by the user. And while the user may place considerable trust into their own hardware, this hardware may not be trustworthy enough for the Web application in regard to non-repudiability requirements.

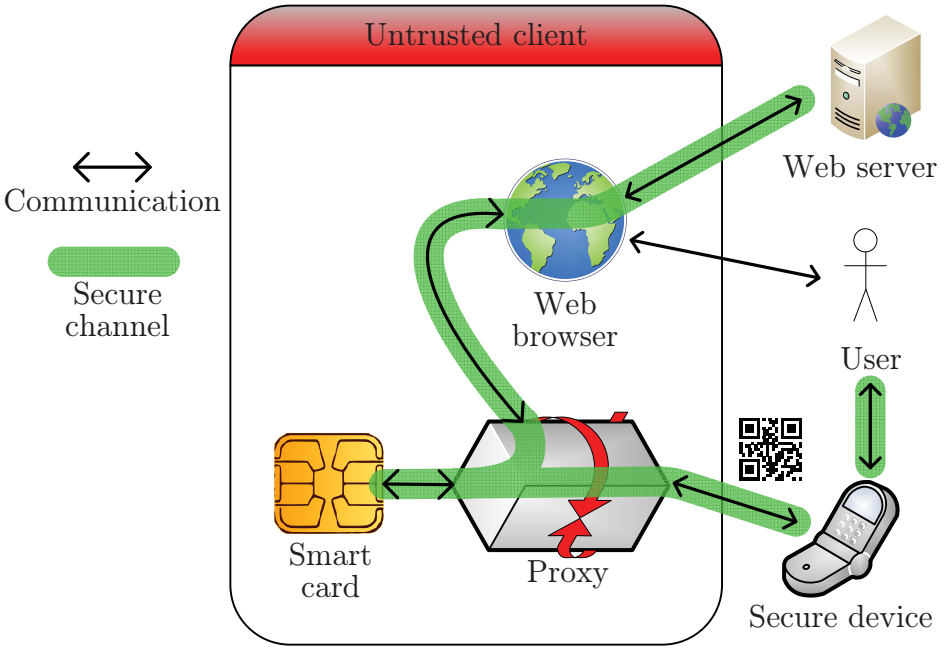
An overview of our architecture is given in Figure 6.1, which illustrates the major components and communication paths, but not the sequence of interactions detailed later. Figure 6.1(a) shows the architecture when used in combination with TPM and Figure 6.1(b) shows the architecture when used in combination with QR-TAN. The black arrows indicate direct communication paths between two entities while the highlighted broader lines in the background depict the secure channels in our system. If a secure channel spans several black arrows, this means that the intermediate entities are untrusted and data are passed through that entities by means of encryption or digital signatures. The trust relations between the constituents described in the following paragraphs are depicted in Figure 6.2. Arrows labelled "high" indicate that a component is highly trusted, while "partial" indicates a lower trust relationship.

Web application and Web server. The Web application is an entity that wants to interact with the smart card; for example a banking site that requires a digital signature before conducting a transaction. The user trusts the Web application for communication with the smart card. However, the user does not trust the Web application with unrestricted access to their computer—e.g., to execute binary code obtained from the Web application. The server-side part of the Web application is running on the Web server, while the client-side part of the Web application is implemented in JavaScript and running on the Web browser. The term "Web application" refers to the combination of these two components.

Web browser. The Web browser is the entity used to interact with the smart card. It hosts the client-side part of the Web application and interacts with the server-side part of the Web application and the smart card. The



(a) TPM



(b) QR-TAN

Figure 6.1: System components

user needs to trust the Web browser for the type of executed transaction. For low security transactions such as reading a stored-value counter, a normal Web browser can be used. For high security transactions, the trust in the Web browser can either be increased by executing the Web browser inside a *trusted environment* (see Section 6.4), or the trust requirements in the Web browser can be decreased by outsourcing part of the transaction to a trusted secure device (see Section 6.5).

Proxy. The proxy is our application responsible for mapping requests from a Web browser to a smart card. Combined with our generic mapping approach (Section 6.3), only a single generic proxy provided by a trusted vendor is required to be installed in order to allow access to smart cards from a multitude of authorized Web applications using state-of-the-art Web technologies. The trust requirements in the proxy are two-fold: From a user's perspective, the proxy is running on a semi-trusted platform as the proxy is started on their local operating system. However, from a smart card issuers perspective the proxy is not necessarily trusted, as malware could control the computer. Thus, the smart card issuer can mandate additional security measures such as the authentication over a TPM (Section 6.4) or QR-TANs [4] (Section 6.5).

Smart card. A smart card is issued by an entity such as a local bank and is responsible for signing sensible data or for executing sensitive transactions. User and Web application have trust in the smart card's correctness. However, the user does not have a direct input and output path to the smart card. Thus, malware can manipulate the user's communication with the smart card.

6.3 Generic proxy and mapping approach

Our generic proxy Web-enables smart cards without installation of custom on-card software and hence is applicable to a large range of existing smart cards. In sections 6.4 and 6.5, we enhance our approach to provide even better security in particular against malware for cases where the on-card software can be adapted.

The developed mapping technique uses a proxy to provide Web applications with access to smart cards and—in addition—protect smart cards from malicious Web applications. In contrast to existing techniques, our *generic mapping* allows a single executable trusted by the user to be used for a diverse set of Web applications and smart cards, not requiring the user to

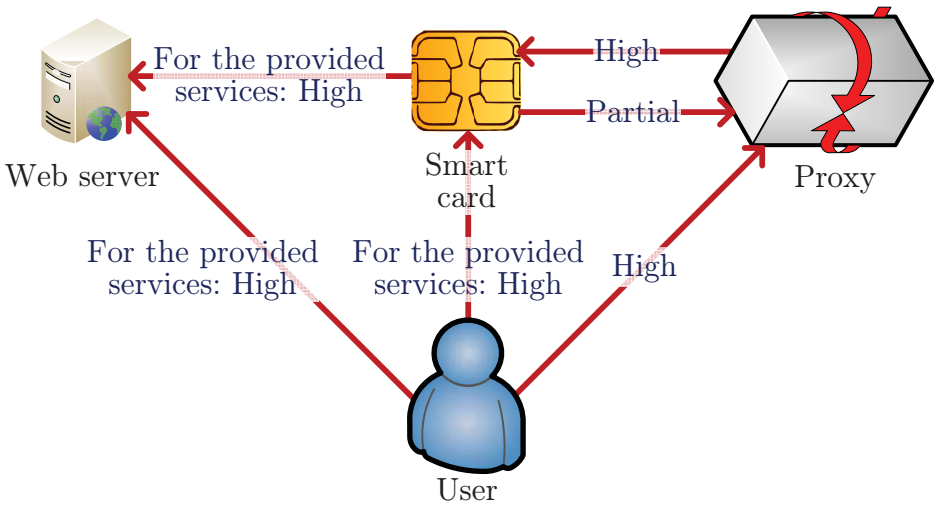


Figure 6.2: Trust relations

trust and execute code obtained from many different Web sites. We validated our concepts with prototype implementations, showing that (i) the proxy concept is feasible in practice, (ii) correctly serves as a filter between Web applications and smart cards, and (iii) allows Web access to smart cards using state-of-the-art Web technologies.

Our system uses a mapping configuration to map abstract method calls to APDUs; an example is shown in Figure 6.3. This configuration defines how invoked methods and their arguments are mapped to APDUs and how results are mapped back to a structure. Furthermore, it includes a list of trusted origins, defining Web sites that may use the mapping, and a list of ATRs (Answer To Resets)—identifying accessible cards. An AID (Application Identifier) identifies the respective smart card application. The mapping is cryptographically signed by the card issuer with a key certified by a Privacy CA (Certificate Authority).

Mapping procedure. The following steps detail how a Web application can call a particular method defined in the mapping file. The interaction between the different components is shown in Figure 6.4.

1. The Web browser obtains a mapping definition and transmits the mapping to the proxy via an RPC (Remote Procedure Call) call by using JavaScript.

```
<mapping>
  <smartcard atr="3b134028351180" aid="A000000062030l0C0202" />
  <method name="login">
    <request>
      <args><arg name="pin" type="STRING" /></args>
      <apdu-mapping is="D4"><argument name="pin" /></apdu-mapping>
    </request>
    <response />
  </method>
</mapping>
```

Figure 6.3: Request mapping example

2. The proxy verifies that the origin of the client-side Web application part running in the Web browser matches the origin defined in the mapping file and verifies the signature of the mapping. To verify the origin the proxy can provide a secret to the Web application's callback URL defined in the mapping file that the Web application needs to include in subsequent requests to the proxy.

3. The proxy verifies that there is a card in the reader and that the ATR of the card matches the ATR of the mapping. If the ATR is different, or if during the remaining process the ATR changes (e.g., because the card is replaced), the proxy will reset.

4. The proxy either asks the smart card if the public key used to sign the mapping should be trusted, or—alternatively—only verifies if the public key has been signed by a trusted certificate authority. In both cases the process is continued. Otherwise, the process is aborted.

5. The proxy receives RPC requests from the Web application's JavaScript code running inside the browser and converts them to APDUs according to the mapping. These APDUs are subsequently transmitted to the smart card. After the response APDU is received from the smart card it is converted and sent back to the application running in the Web browser.

Summarized, to protect smart cards from malicious Web applications we first identify the type of smart card connected to the PC. We proceed by verifying if the smart card provider has authorized the mapping file[1] provided by the Web

[1]The authorization of a Web application's mapping file is an administrative task in contrast to the development and installation of on-card software, which would require re-distribution of smart cards.

application for this particular type of smart card. If this verification succeeds, this mapping is then used to restrict which Web applications can access the smart card and to restrict the type of APDUs that the Web application may transmit to the smart card.

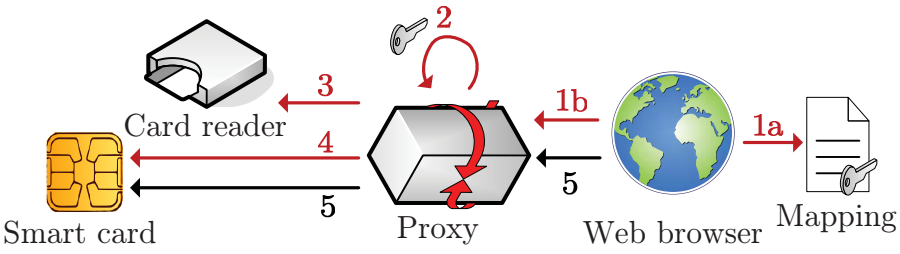


Figure 6.4: Mapping procedure

6.4 Smart card-based TPM attestation

This and the next section show extensions for establishment of a secure channel between the smart card and either (i) the Web browser running in a secure environment, or (ii) a mobile device trusted by the user. The primary motivation for these extensions is to mitigate attacks by adversaries with access to the user's computer. Both extensions require the support of on-card software.

The first approach discussed in this section uses an end-to-end security protocol between the smart card and the Web browser that provides authentication, integrity and confidentiality between the two endpoints. Our protocol works on a layer between APDU transmission and APDU interpretation. Conceptually, it can be compared to TLS (Transport Layer Security). However, instead of desktop computers it targets smart cards and uses the remote attestation features of TPM that allow a remote party to verify if a computer is running a particular software configuration. The main requirements of our protocol are: (i) *Authentication*: Each party must be able to securely authenticate the other party. (ii) *Integrity*: Each party must be able to verify that the transmitted data has not been manipulated. (iii) *Confidentiality* (optionally): End-to-end encryption between the parties must be possible.

The first two items are required to prevent manipulation of transmitted data: Without authentication of the remote party, an attacker could directly establish a connection to one of the parties. Furthermore, without integrity

of individual data items, an attacker could use a man-in-the-middle attack to manipulate these data items while in transit. The third item is optional: Without encryption, a man-in-the-middle is able to read transmitted information, but not able to manipulate this information. By using encryption, we can prevent an attacker from learning information about ongoing transactions.

In the following sections we first introduce the TPM functionality we use for remote attestation. Afterwards, we continue with a description of our secure channel that provides authentication, integrity and confidentiality. While a secure channel is already part of the Global Platform specification[2], the specification assumes that there is a shared secret key between smart card and accessor. However, as we want to enable access to smart cards from many different Web sites, a shared secret between the smart card and each individual Web site is infeasible.

6.4.1 Secure computer model

For the endpoint of our end-to-end security protocol on the local computer (see Figure 6.1(a)) we assume a computer model that allows to create a *secure runtime partition* in which software is executed that cannot be accessed or manipulated by the user's (default) operating system. This secure partition hosts a browser instance used for communication with the smart card. Furthermore, the secure partition allows for remote attestation—allowing a remote entity to securely identify the executed software. To provide compatibility across different types of trusted environments, our model does not assume any further characteristics. In particular, we do *not* assume that it is possible to open a secure channel to I/O (Input/Output) devices such as smart card readers. This is in accordance with the current state of trusted environments, where applications can open secure channels only to some types of I/O devices such as monitors and keyboards [Mül08b, CYC+07].

There are different technologies that allow for the creation of such a secure partition. One technology is Intel's TXT (Trusted Execution Technology) that complements the functionality of a TPM by allowing a secure hypervisor to provide *virtual environments* that are protected from access by malicious applications. For remote attestation a TPM can be used. The Xen hypervisor provides a *vTPM* [BCG+06] implementation that provides *virtual* TPM chips [EL08] to the executed instances. These virtual TPMs use features of the host's hardware TPM for the secure implementation of their functionality.

[2] `http://www.globalplatform.org/`

6.4.2 Establishing a shared secret for HMAC and encryption

As basis for encryption and authentication we use a shared secret between smart card and Web application running in a trusted environment. To establish this secret we use an authenticated DH (Diffie-Hellman) key exchange [DH76] as depicted in Figure 6.5. The variables g, p, A, B in the figure are Diffie-Hellman parameters. For authentication, we sign the parameter set sent by each of the parties with digital signatures that prevent man-in-the-middle attacks. On the smart card we use an asymmetric key that is certified by the smart card manufacturer, while on the TPM we use the AIK (Attestation Identity Key) for authentication.

Instead of using Diffie-Hellman, it would also be possible to generate the symmetric key on one of the endpoints and use asymmetric encryption to transfer this key to the other endpoint. However, with such a method the long-term security of the communication would depend on the security of the particular asymmetric key. If the asymmetric key would be broken, each symmetric key encrypted with this key in the past would be compromised. With Diffie-Hellman on the other hand, an attacker needs to crack each session key individually.

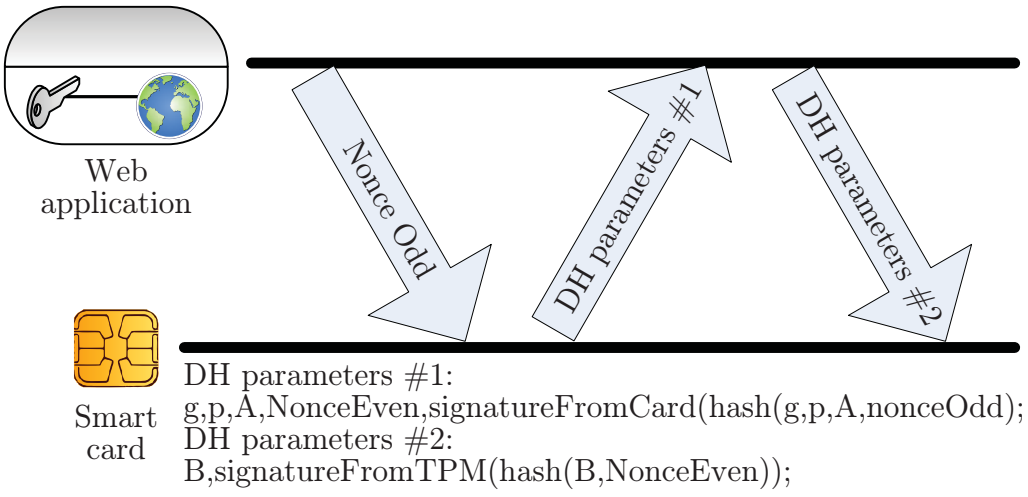


Figure 6.5: Exchange of Diffie-Hellman parameters for secure channel. A nonce is used to prevent replay attacks. Each endpoint transfers cryptographically signed DH parameters to the other endpoint. These parameters are then used to establish the key for the secure channel.

6.4.3 Mutual authentication and integrity

Mutual authentication allows each endpoint of a conversation to authenticate the identity of the opposing endpoint. Authentication and integrity are intertwined concepts: When endpoint authentication is used without data integrity, an adversary can exchange data while in transit. Likewise, if data integrity is used without endpoint authentication, an endpoint knows that the data have not been modified, but does not know the identity of the remote endpoint.

In this section we provide our approach for authentication and integrity. There are two endpoints (see Figure 6.1(a)): The smart card and the Web browser running in a trusted environment. Authentication uses authenticated Diffie-Hellman key exchange, while integrity uses HMACs (Hash-based Message Authentication Codes).

Each smart card stores a custom key pair generated on initialization and digitally signed by the smart card issuer. When transmitting Diffie-Hellman parameters, the smart card signs them with its private key and appends the public key together with the certificate of the issuer to the data structure.

On the PC, the `TPM_Quote` command of the TPM is used to sign the content of particular PCRs (Platform Configuration Registers) that contain measurements of the executed software used to identify a particular software configuration. When PCRs are updated, the TPM combines the existing value with the new value, thus software running on the computer is not able to set the registers to arbitrary values. By cryptographically signing the values within the registers, the TPM chip can attest the state of the system to a remote system. This functionality is also called *remote attestation*.

The certification of the TPM's key is more complex than in the case of the smart card. Figure 6.6 shows the certification and attestation procedure, illustrating which entity certifies which other entity to build up a chain of trust that allows the smart card to verify a correct software execution environment. The TPM contains two different types of keys: The unmodifiable EK (Endorsement Key) generated during production and certified by the TPM's manufacturer and a modifiable AIK used for attestation. A Privacy CA with knowledge about EK and AIK is responsible for certifying the AIK [Mül08b, CYC+07]. During attestation, the AIK signs a set of values that identifies the executed software. The smart card compares these values with a set of reference values identifying a particular browser appliance and signed by a trusted party (e.g., the smart card issuer).

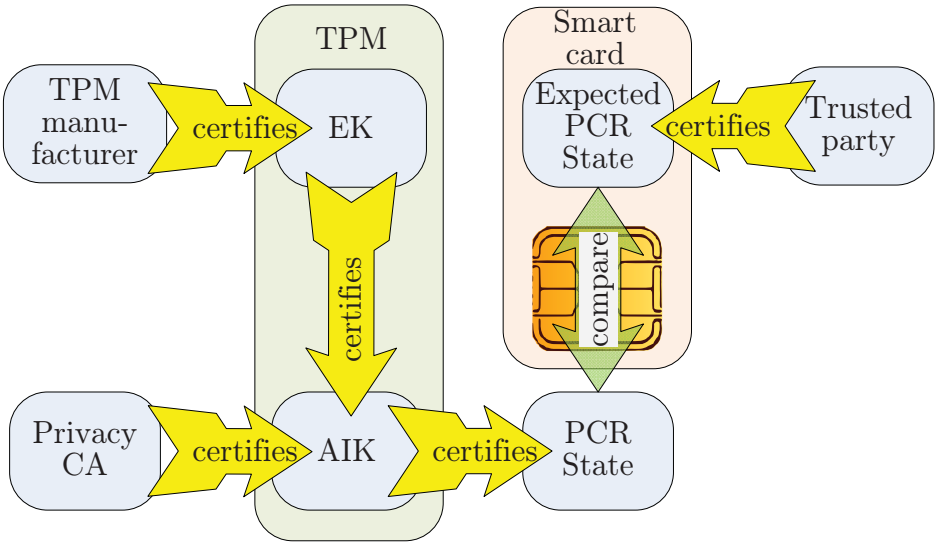


Figure 6.6: Certification and attestation procedure

6.4.4 APDU encryption and authentication

For encryption and authentication of APDUs we use a protocol similar to smart card *secure messaging* defined in ISO (International Organization for Standardization)/IEC (International Electrotechnical Commission) 7816-4. The main reason why we cannot directly use secure messaging is that secure messaging requires a shared key between smart card and terminal. However, in our application scenario, this is not feasible as we want to enable secure communication with the smart card from a wide range of Web applications.

In theory, it would be possible to use the secret we established in Section 6.4.2 as the basis of a ISO/IEC 7816-4 compliant communication established directly by the smart card's operating system. However, as common smart card operating systems do not allow for access to the particular layer by client applications, this option is not possible. Instead, we re-implement comparable functionality inside the application layer, transmitting secured data as payload in APDUs. Therefore, APDU encryption and authentication allows us to establish a secure channel between smart card and Web browser.

APDUs are encrypted and authenticated by calculating *encrypt(session_key, hmac(session_key, orig_apdu + counter) + orig_apdu + counter)* using a symmetric encryption protocol such as AES. If no encryption is required, using an HMAC without encryption is possible. The HMAC serves to detect manipulation attempts of the APDU. The counter allows to detect replay

attacks: In the beginning, a counter value derived from the shared key is used. As the session proceeds, each endpoint increases the counter value by one for each request and each response. Furthermore, each endpoint can detect if the received counter value matches the expected counter value. As a side effect, the counter also acts as a type of IV (Initialization Vector), as two equal APDUs encrypt to two different cipher texts.

6.4.5 Security discussion

One issue with the certification of the PCR state by the smart card is that if the Privacy CA only certifies that the AIK belongs to *any* valid TPM implementation, it would be sufficient for an adversary to obtain the private key of *any* certified TPM to apply signatures. To cause a security problem the adversary would need to (i) break the user's environment so that the browser does not run inside a secure environment with the effect that malware has access to the software and (ii) to sign the TPM's side of the transaction with a certified key from another broken TPM implementation.

One option for the mitigation of such an issue would be for the Privacy CA to not only certify that the key belongs to any TPM implementation, but to further include a user identifier in the certificate. This certificate can then be used by the smart card to verify that the TPM belongs to an authorized user. Another option is to remember the first TPM key used for authentication and to only allow this particular key for future transactions. While this does not help against malware on a freshly installed PC, it protects the user against later attacks. However, to use another PC, a user would first need to obtain a certificate from the card issuer that instructs the smart card to reset the stored key.

When a secure channel is used, there is an important difference in the behavior of the intermediate smart card proxy: In unencrypted communication, the proxy is responsible for filtering the requests, i.e., to only allow requests *whitelisted* in the mapping to pass. However, when encryption is used, such a filtering is not possible as the proxy does not have access to the plain text. Thus, the proxy cannot verify if a particular encrypted APDU is allowed by the mapping file. As the proxy cannot filter requests sent to smart cards in that case, smart cards need to be developed with the assumption that potentially *any* Web site can send requests. While a majority of smart cards is already designed to withstand external attacks, the optimal mitigation strategies in our scenario are different. Traditionally, a smart card does, e.g., deactivate itself, if large amounts of failed authentication attempts are

detected. However, if any Web application can communicate with the smart card, a Web application could abuse such a behavior for a DOS (Denial Of Service) attack: By deliberately causing failed authentication attempts, any Web application could disable the smart card.

As mitigation strategy, smart cards should not take any *destructive* actions in case of failed authentication attempts or other types of security alerts that can be caused by external Web applications. For example, instead of deactivating the smart card in case of multiple failed authentication attempts the smart card could just increase the minimum interval required between each authentication attempt. As an alternative, the on-card application responsible for the secure connection can filter requests according to the information in the mapping—and thereby accomplish the filtering task of the proxy. However, the drawback of this approach is that it leads to a higher complexity of the on-card application.

6.5 Authentication with QR-TAN

This section presents the second option to increase the security of the proxy, by extending our system with our QR-TAN protocol discussed in Chapter 5. In comparison to our original QR-TAN approach [4], our modifications allow the use of QR-TANs without the interaction of a server.

Conceptually, it is sufficient to replace the RTC (Remote Trusted Computer) in the original approach with a smart card. However, due to the different capabilities of smart cards and servers, modifications to the original approach allow for better integration. In particular, our modifications address the following issues:

1. On smart cards, the generation of textual authentication requests intended for humans is more complicated than on servers. Especially as the smart card's memory restricts the amount of stored localizations and as it is rather complex to update the messages once the card has been issued.

2. The smart card should have the capability to decide if external transaction authentication is required. For example, in banking applications a smart card may allow daily transactions of up to a particular total value without authentication, only requiring authentication above that value.

3. Usage of QR-TAN should be transparent to applications using the proxy. Thus, only the smart card, the proxy, and the secure device should contain QR-TAN specific code.

To enable these properties, we extend our mapping description to contain information about QR-TAN authentication. In particular, we introduce a new `<auth />` section that describes which status words in the APDU response indicate that QR-TAN authentication is required and how human readable text is generated from the structure returned by the smart card. For authentication the following steps depicted in Figure 6.7 are used:

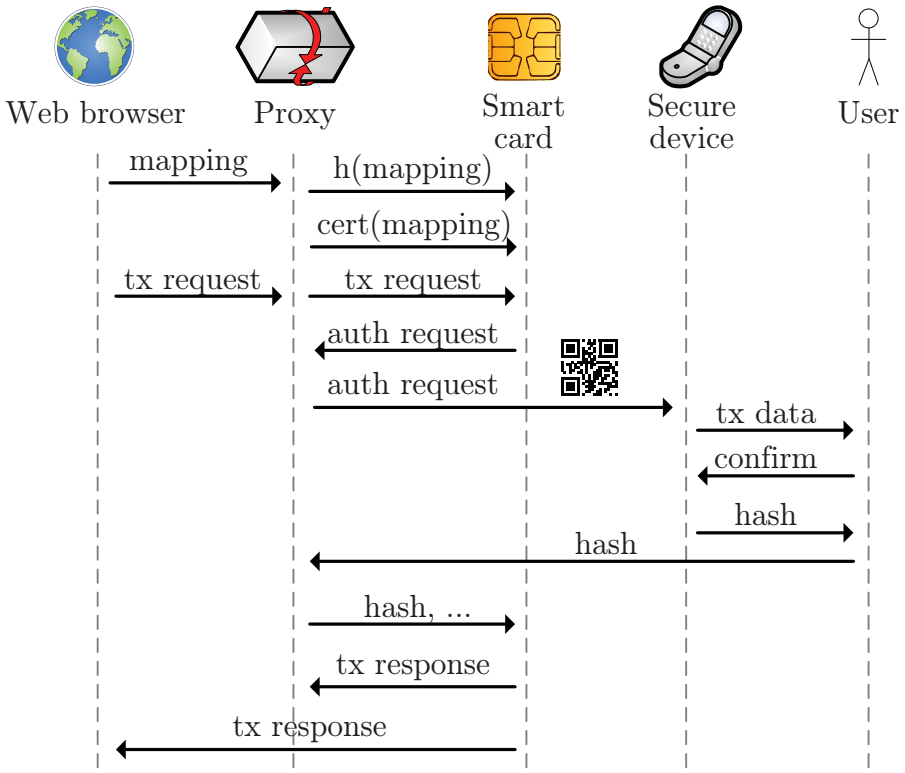


Figure 6.7: QR-TAN authentication steps

1. On initialization the browser provides the mapping to the proxy. The proxy transmits a hash and a certificate of the used mapping file signed by a trusted third party to the smart card. The smart card verifies that the certificate allows use of the given mapping file.

2. The Web browser sends its transaction request to the smart card.

3. The smart card responds with a particular status word indicating that QR-TAN authentication is required for this type of transaction. The data of the response contains a structure with information about the transaction, a nonce, and the secure hash of the used mapping file.

4. The proxy reads the response by the smart card and converts it to a QR code that is subsequently displayed to and scanned by the user's secure device.

5. The secure device first checks if it has stored the mapping file indicated in the QR code. If not, it prompts the user to install the mapping file—e.g., by scanning a compressed QR code containing the mapping file. Otherwise, it generates a human readable text according to the information in the mapping file and asks the user for confirmation.

6. If the user confirms, the secure device generates a shortened hash based on the data structure of the original request (in step 2), the nonce, and the hash of the mapping file and presents this shortened hash to the user.

7. When the user provides this shortened hash to the proxy, it generates an APDU with this information and sends it together with the nonce and the hash of the mapping file to the smart card. The smart card validates if the locally computed hash for transaction matches the shortened hash within the APDU.

8. In case of success, the smart card returns the response of the original APDU request issued in step 2.

The conversion of the transaction data to a human readable text can be done on either one of the two endpoints of the QR-TAN authentication: Inside the smart card or inside the trusted secure device. It is not possible to do this conversion on any device between these two endpoints as this would prevent successful end-to-end authentication. In our approach we perform this conversion inside the secure device. While generating the text directly within the smart card would be conceptually simpler, it would require the smart card to store potentially large amounts of textual data—e.g., if multiple localizations are required. Furthermore, it is not easily possible to adapt the text once the smart card has been issued.

As the secure device uses a mapping file to convert the structure with information about the transaction to a textual format, it must ensure that the mapping file is also authenticated. Otherwise, an attacker would be able to

send a manipulated mapping file to the secure device, causing the device to show incorrect transaction information to the user. It is not sufficient for the secure device to only validate if the mapping file has been signed by a trusted party: As the secure device cannot securely obtain the ATR of the smart card, it cannot ensure that the mapping file belongs to a particular smart card. Instead, we include a hash of the mapping file in the structure that is hashed by the secure device, allowing the smart card to ensure that the QR-TAN belongs to a particular mapping. While the smart card does not need to know the content of the mapping file, it needs to know the digital signature to decide if it can trust the mapping. Thus, the proxy can send the signature of a mapping to the smart card via APDUs.

By integrating our QR-TAN approach directly with the proxy, the proxy is responsible for displaying the QR code and for forwarding the QR-TAN back to the smart card. Thus, the whole process can be transparent to applications using the mapping, as the only difference between authentication and non-authentication is the additional delay of the authentication process.

6.6 Related work

This section describes related work to Web-enabled smart cards as provided by our smart card proxy. Work related to the QR-TAN extension of our proxy is discussed in context of QR-TAN in Section 5.6.

Itoi et al. [IFH00] describe an approach for secure Internet smart cards that allows users to access remote smart cards over the Internet. In contrast, we provide the client-side part of a Web application running in a Web browser with access to smart cards at the local computer. Thus, the security assumptions and implementation details differ fundamentally.

An expired IETF (Internet Engineering Task Force) Internet draft for SmartTP by P. Urien [Uri01] specifies a unique software stack applicable to different types of smart cards, but—unlike our approach—requires software support from the smart card. Hence, it is not applicable to legacy cards.

The TLS-Tandem approach [Uri09] seems to use smart cards for access control to a Web server, while we aim at Web-enabling smart cards to mitigate man-in-the-middle attacks.

The OMA (Open Mobile Alliance) SCWS (Smartcard Webserver) specification defines the interface to an HTTP server on a smart card. However, compared to our proxy such an approach is not directly applicable to desktop operating

systems as the specification is targeted against mobile devices with smart cards used as USIM (Universal Subscriber Identification Module), UICC (Universal Integrated Circuit Card), or R-UIM (Removable User Identification Module). As a consequence, no device drivers that provide SCWS-based HTTP support to desktop operating systems are available.

In addition to the works discussed above, a comprehensive examination of approaches to provide smart cards with network access is provided by HongQian [Lu07].

6.7 Conclusions

In this chapter we presented a secure approach for Web-based smart card communication. Our overall contributions are: (i) a secure technique using a single generic proxy to allow a multitude of authorized Web applications to communicate with *existing* smart cards and (ii) techniques for *new* smart cards that allow for secure end-to-end communication between a user and a smart card. In particular, our security extensions cover the usage of (i) a TPM and (ii) QR-TANs to secure communication with smart cards.

Compared to related approaches, our system works with existing smart cards without requiring changes to on-card software. Thereby, we can increase the security of the user's system, by not requiring the user to install privileged software distributed by Web sites that require access to a smart card. Furthermore, in cases where it is feasible to adapt on-card software, we can increase the security over the state-of-the-art even further, as we can use the TPM or QR-TANs to secure transactions that would otherwise be affected by malware on the terminal.

Chapter 7

Conclusions and future work

In this chapter we first discuss alternative application scenarios in Section 7.1. We then proceed to our conclusions regarding the contributions of this thesis in Section 7.2. Finally, we discuss future work in Section 7.3.

7.1 Alternative application scenarios

This section discusses alternative application scenarios for our contributions with the discussion into the following three areas: (i) Adaptive rate control and performance, (ii) time synchronization and timestamping, and (iii) secure communication and transaction authentication.

7.1.1 Adaptive rate control and performance

In theory our adaptive rate control approach is applicable to all application scenarios where it is essential to control the request rates of a set of clients, such as Web applications. However, in practice the applicability of our approach is limited due to the required protocol adaptations, as such changes are not compatible with all types of existing protocols. In addition, the feasibility of the required adaptations also depends on the fact whether a piggyback approach is sufficient, or a distributed feedback channel is required.

A rather general application scenario are Web services, if those services allow requests to be delayed. In such cases our adaptive rate control approach can mitigate peak loads on the server by buffering requests directly on the client, if the server's load is too high. If the original time of a request is not essential,

or if all participants of such a service are operated by entities fully trusting each other, our adaptive rate control approach can be implemented without our smart card based timestamping protocol.

As an example of potential benefits possible with our temporal decoupling approach, consider the talk of Google engineer Luiz André Barroso at the O'Reilly Velocity 2008 conference[1] who stated that Google's servers are dormant most of the time with only occasional spikes of peak activity. The primary problem in such cases is that servers need to be dimensioned for peak loads, instead of average loads. Our temporal decoupling approach allows to reduce peak loads in certain application scenarios and would therefore potentially allow to reduce the amount of required servers.

Furthermore, our distributed feedback channel can complement related work [GSTBU10], as it allows to efficiently broadcast client request rate settings to a set of clients. This decreases the load of the server, as the server does not need to reject requests by individual clients anymore, and thereby also increases the amount of clients a particular server can handle.

7.1.2 Time synchronization and timestamping

Our main contributions in this area are interval-based time synchronization, smart card based time synchronization, and distributed timestamping.

One potential application scenario are distributed computer games, where a major issue is to prevent players from cheating. In a fully distributed game where player's exchange actions with each other, individual players can gain an unfair advantage by holding back the transmission of their own actions until they have received the actions of the other players. In such cases malicious players can react to the actions they receive by transmitting their own actions with backdated timestamps. While today's protocols such as the Lockstep protocol [BL01] can mitigate this problem, they exhibit a higher latency than simple approaches where players directly exchange messages. With our secure smart card and interval-based timestamping approaches it would be possible to enforce client-side timestamps for actions before they are relayed to other players. While this does not prevent malicious players from delaying transmission of their actions, they cannot backdate their own actions anymore, and thus do not gain an essential advantage over other players.

Another application scenario are electronic transactions, such as processed by banks. In such cases there may be a regulatory agency that needs to be

[1] see `http://velocityconf.com/velocity2008/public/schedule/detail/3694`

able to verify the validity of transactions without receiving a copy of each individual transaction. For example, verification can be beneficial if there is a dispute over a transaction between the bank and a customer. With our secure timestamping approach the regulatory agency can issue smart cards that are used to locally timestamp transactions at the individual banks. In case of disputes, a bank can prove that information about a particular transaction existed at a given point in time. While this does not prevent a bank from creating a forged transaction, it prevents the bank from retroactively tampering with transactions. A related scenario are log files on a server, where a secure timestamping approach can prevent potential intruders from retroactively manipulating information within a log file.

While both application scenarios can be implemented using our smart card based timestamping approach, their security can be increased with our distributed timestamping protocol, which prevents adversaries from successfully using physical attacks against a single smart card to manipulate timestamps.

7.1.3 Secure communication and transaction authentication

Our main contributions in the area of secure communication and transaction authentication are our QR-TAN transaction authentication protocol and our smart card proxy.

QR-TAN is applicable to any type of service that requires secure transaction authentication. Examples are banking transactions, where QR-TANs can be used instead of traditional TAN codes. However, as QR-TAN software can be deployed to standard smartphones, QR-TANs can also be used in application scenarios where the distribution of traditional TAN codes or mobile TANs would be too expensive, such as Web sites that require an increased security of their login process. The demand for such secure user authentication is shown by the fact that service providers such as Google recently started to offer optional two-factor authentication for some of their services. The main advantage of QR-TANs over existing solutions is that QR-TANs can be used with any smartphone that includes a camera. In addition, use of QR-TANs does not require an Internet connection. Thus, QR-TAN authentication can also be used when traveling at locations where no data-roaming is available.

Our smart card proxy is applicable to all types of existing smart card applications, where access to the smart card is required by a Web application. Examples of such services are citizen card applications and electronic banking

transactions secured with smart cards. Due to the characteristics of our protocol, our proxy can be used for existing services without requiring changes to on-card applications.

7.2 Conclusions

In this thesis we contributed with techniques to increase the performance, availability, and security of first-price sealed-bid online auctions. First-price sealed-bid auction [MM87] scenarios generally exhibit high peak loads around the auction deadline as a majority of bidders tries to submit bids shortly before the deadline. Moreover, these auctions also exhibit high dependability requirements as an auction canceled due to technical reasons can lead to significant financial losses, because of market conditions changing over time. As a consequence, systems need to be designed for excessive workloads and high availability, leading to massive over-provisioning and high costs.

In addition to our temporal decoupling techniques for first-price sealed-bid auctions, this thesis deals with security issues in time synchronization and timestamping, as well as with security issues in transaction authentication and smart card communication. In particular, the contributions of this thesis are split into the following three main areas:

- In the area of adaptive load control and performance for first-price sealed-bid auctions our main contribution is the integration of (i) a distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain client requests directly at the client-side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization.

- In the area of time synchronization and timestamping we first contributed with a secure time synchronization and timestamping protocol for resource-constrained devices to enable secure timestamps to be applied within smart cards, for example. Moreover, we showed the feasibility of our approach based on a .NET smart card implementation. In addition, we presented a smart card based approach for distributed timestamping that is able to provide low latencies due to connection pre-establishment.

- In the area of security we contributed with the QR-TAN technique for secure transaction authentication and a smart card proxy that allows

Web applications to communicate with smart cards. In addition, we presented two techniques for secure end-to-end communication between the user and the smart card.

Key idea of the approach presented in this thesis is to alleviate the dependability problems by shifting them into the security domain and by consequently solving the new security problems. We increase dependability of the system by temporally decoupling the individual components of the system from each other, and allowing users to place bids on trusted devices physically located at their place. However, by doing so we decrease the security of the system, as we give adversaries new options to attack the system.

Therefore, we subsequently solve the security problems: Software running on clients cannot be protected against attacks by malicious users. As a consequence, we introduced a smart card based time synchronization and timestamping protocol that is able to mitigate asynchronous delays and delay attacks. However, even smart cards do not necessarily provide adequate security against physical attacks. Therefore, we extended our timestamping protocol to work in a distributed mode, in order to split the trust requirements amongst several smart cards at different physical locations.

Using smart cards requires a secure mechanism for communication between a Web application and a smart card. Current state-of-the art does not allow for this type of secure communication, as it typically requires the user to fully trust executable code obtained by each individual service provider. Instead, our approach allows generic Web applications to access generic smart cards. We then increase the security of this approach by introducing QR-TANs for transaction authentication and TPM based transactions for malware mitigation.

7.3 Future work

Peak load reduction is a major issue, in today's cloud deployments, especially due to requirements such as energy efficiency. Traditional solutions to decrease such peak loads primarily deal with load distribution amongst servers, for example by scheduling transactions in such a way that the load is equally spread within a cluster. Our temporal decoupling technique is a complementary approach that spreads the load of individual transactions in the temporal domain. While only applicable to application scenarios that can be temporally decoupled, the solution approach is fundamentally different from existing work. Consequently, a goal for future work is to identify

how to combine our temporal decoupling approach with existing solutions. This would not only allow to increase the efficiency of certain types of cloud deployments, but does potentially also allow to improve the performance of our decoupling approach when deployed in a cluster setup. In addition, the performance of the controller is only suboptimal when a *piggyback* approach is used. Therefore, a further goal for future work is to use a Smith predictor [SO03, ZLM07] to compensate for dead time due to delays and thereby stabilize the controller as well as an evaluation of fuzzy controllers [CC07] to verify if they are able to yield acceptable results in our system.

In the area of time synchronization NTP is currently the standard protocol. However, the possible accuracy of NTP is limited in case of asynchronous delays. While our time synchronization and timestamping protocol discussed in Chapter 3 can improve on the state-of-the-art for certain application scenarios, it is currently not optimized for cases where determining a single point in time is more important than being able to determine an interval containing the correct time. Consequently, adaption of our time synchronization and timestamping protocols for a wider range of application scenarios is one future research goal. For example, in distributed computer games such a timestamping approach would allow to improve on the performance of the Lockstep protocol [BL01, MKY+07]. However, adaption of our protocol would require performance optimizations, to prevent high amounts of actions on each client to overload the smart card.

In comparison to related work our distributed timestamping protocol discussed in Chapter 4 can deterministically choose nodes responsible for timestamping, provided that the nodes in the network are able to reach an approximate agreement on the network size. While we outline techniques on how such an agreement can be reached, those techniques partly depend on central components within the network. To increase the availability of our protocol, it would be beneficial to minimize dependence on such central components. In addition, while hop-to-hop latencies are amplified by our overlay approach, this can be mitigated with techniques such as Pharos [CXS+07] that allow to find nodes with low latencies. Furthermore, our protocol currently provides semi-anonymity, but there are no formal security guarantees yet. As future work we therefore identify an examination on how to increase the provided level of anonymity and a subsequent definition of formal security guarantees.

In the area of secure communication and transaction authentication our QR-TAN protocol allows for secure transaction authentication. While QR-TANs are designed to work with any type of secure device capable to scan QR codes, our primary application scenario are smart phones. As a consequence, there

are various approaches that can be used to further increase the security of QR-TANs against adversaries. For example, JSR-177 (Security and Trust Services API for J2ME) defines a secure storage element to protect sensitive data. When used in combination with QR-TAN this would allow to protect the shared secrets used by the protocol from adversaries with physical access to the phone.

In regard to our smart card proxy—which constitutes our second contribution in this area—our general approach of Web-enabling smart cards is also applicable to other areas that do not use APDUs. For example, in the area of home automation our proxy approach can be used to Web-enable legacy devices. As in the case of smart cards, security characteristics are essential to prevent adversaries from tampering with the functionality of such devices. As future work we identify the combination with state-of-the-art approaches for automatic generation of network protocol gateways [BRLM09], as this would allow for more efficient generation of Web to smart card mapping files. In addition, recent developments such as the TEM (Trusted Execution Module) [CSvDD08] will allow for the implementation of more powerful on-card request mapping approaches.

Bibliography

[AdC+09] Jean-Daniel Aussel, Jerome d'Annoville, Laurent Castillo, Stephane Durand, Thierry Fabre, Karen Lu, and Asad Ali. Smart cards and remote entrusting. In *Future of Trust in Computing*, pages 38–45. Vieweg+Teubner, 2009.

[AEH75] E. A. Akkoyunlu, K. Ekanandham, and R. V. Huber. Some constraints and tradeoffs in the design of network communications. In *SOSP*, pages 67–74, 1975.

[ASB02] Tarek F. Abdelzaher, Kang G. Shin, and Nina T. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002.

[BBK03] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer, 2003.

[BCG+06] Stefan Berger, Ramon Caceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th USENIX Security Symposium*, pages 305–320. USENIX, August 2006.

[BD07] Andrea Bottoni and Gianluca Dini. Improving authentication of remote card transactions with mobile personal trusted devices. *Computer Communications, Elsevier*, 30(8):1697–1712, 2007.

[BF99] Dirk Balfanz and Edward W. Felten. Hand-held computers can be better smart cards. In *SSYM'99: Proceedings of the 8th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 1999. USENIX Association.

[Bis90] M. Bishop. A security analysis of the NTP protocol version 2. In *Computer Security Applications Conference, 1990., Proceedings of the Sixth Annual*, pages 20–29, Tucson, AZ, USA, December 1990.

[BL01] Nathaniel E. Baughman and Brian Neil Levine. Cheat-proof playout for centralized and distributed online games. In *INFOCOM*, pages 104–113, 2001.

[BLGB06] A. Bonnecaze, P. Liardet, A. Gabillon, and K. Blibech. Secure time-stamping schemes: A distributed point of view. In *Annals of Telecommunications*, volume 61, pages 662–681. Institut Télécom, 2006.

[BR08] Bernd Borchert and Klaus Reinhardt. Device and method for tap-proof and manipulation-proof encoding of online accounts. Patent Application, 12 2008. WO 2009/000223 A9R3.

[BRLM09] Yérom-David Bromberg, Laurent Réveillàre, Julia L. Lawall, and Gilles Muller. Automatic generation of network protocol gateways. In *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 21–41. Springer Berlin / Heidelberg, 2009.

[Bun10] Bundesamt für Sicherheit in der Informationstechnik. Lagebericht 3. Quartal 2010. `https://www.bsi.bund.de/cln_183/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Quartalslagebericht/Quartalslagebericht_3_2010_pdf.html`, December 2010.

[CC07] Ka Ho Chan and Xiaowen Chu. Design of a fuzzy PI controller to guarantee proportional delay differentiation on web servers. In Ming-Yang Kao and Xiang-Yang Li, editors, *AAIM*, volume 4508 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 2007.

[CGK+02] Dwaine E. Clarke, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten van Dijk, Srinivas Devadas, and Ronald L. Rivest. The untrusted computer problem and camera-based authentication. In Friedemann Mattern and Mahmoud Naghshineh, editors, *Pervasive*, volume 2414 of *Lecture Notes in Computer Science*, pages 114–124. Springer, 2002.

[CMZ02] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of ejb applications. In *OOPSLA*, pages 246–261, 2002.

[CSvDD08] Victor Costan, Luis F. G. Sarmenta, Marten van Dijk, and Srinivas Devadas. The trusted execution module: Commodity general-purpose trusted computing. In Gilles Grimaud and François-Xavier Standaert, editors, *CARDIS*, volume 5189 of *Lecture Notes in Computer Science*, pages 133–148. Springer, 2008.

[CVE10a] CVE-2010-1797. National Vulnerability Database, NIST, 2010. `http://nvd.nist.gov/nvd.cfm?cvename=CVE-2010-1797`.

[CVE10b] CVE-2010-2973. National Vulnerability Database, NIST, 2010. `http://nvd.nist.gov/nvd.cfm?cvename=CVE-2010-2973`.

[CXS+07] Yang Chen, Yongqiang Xiong, Xiaohui Shi, Beixing Deng, and Xing Li. Pharos: A decentralized and hierarchical network coordinate system for internet distance prediction. In *GLOBECOM*, pages 421–426. IEEE, 2007.

[CYC+07] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A practical guide to trusted computing*. IBM Press, 2007.

[DH76] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[EL08] Paul England and Jork Löser. Para-virtualized tpm sharing. In Peter Lipp, Ahmad-Reza Sadeghi, and Klaus-Michael Koch, editors, *TRUST*, volume 4968 of *LNCS*, pages 119–132. Springer, 2008.

[FG08] Lorenz Froihofer and Karl M. Goeschka. Balancing of dependability and security in online auctions. In *38th Int. Conference on Dependable Systems and Networks (DSN'08) (Supplementary volume)*, 2008.

[GKŁ06] M. Gogolewski, M. Kutyłowski, and T. Łuczak. Distributed Time-Stamping with Boomerang Onions. *Tatra Mountains Mathematical Publications*, 33:31–40, 2006.

[GSTBU10] Maayan Goldstein, Onn Shehory, Rachel Tzoref-Brill, and Shmuel Ur. Improving throughput via slowdowns. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 11–20, New York, NY, USA, 2010. ACM.

[HÅ02] Tore Hägglund and Karl Johan Åström. Revisiting the Ziegler-Nichols tuning rules for PI control. *Asian Journal of Control*, 4(4):364–380, December 2002.

[HJ99] Manfred Hauswirth and Mehdi Jazayeri. A component and communication model for push systems. In Oscar Nierstrasz and Michel Lemoine, editors, *ESEC / SIGSOFT FSE*, volume 1687 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 1999.

[HKW06] Alain Hiltgen, Thorsten Kramp, and Thomas Weigold. Secure internet banking authentication. *IEEE Security and Privacy*, 4(2):21–29, 2006.

[HOP+86] James O. Henriksen, Robert M. O'Keefe, C. Dennis Pegden, Robert G. Sargent, and Brian W. Unger. Implementations of time (panel). In Douglas W. Jones, editor, *WSC '86: Proceedings of the 18th conference on Winter simulation*, pages 409–416, New York, NY, USA, 1986. ACM.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, Jan. 1991.

[IFH00] Naomaru Itoi, Tomoko Fukuzawa, and Peter Honeyman. Secure internet smartcards. In Isabelle Attali and Thomas P. Jensen, editors, *Java Card Workshop*, volume 2041 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2000.

[JXT] JXTA Project. `https://jxta.dev.java.net/`. (Access date: 29 June, 2010).

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997.

[KBE99] Mieczyslaw M. Kokar, Kenneth Baclawski, and Yonet A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14(3):37–45, 1999.

[KRAW08] Maria Kihl, Anders Robertsson, Anders Andersson, and Björn Wittenmark. Control-theoretic analysis of admission control mechanisms for web server systems. *World Wide Web*, 11(1):93–116, 2008.

[LA04] HongQian Karen Lu and Asad Ali. Prevent online identity theft - using network smart cards for secure online transactions. In Kan Zhang and Yuliang Zheng, editors, *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2004.

[LBG08] Duc-Phong Le, Alexis Bonnecaze, and Alban Gabillon. A secure round-based timestamping scheme with absolute timestamps (short paper). In *ICISS*, volume 5352 of *Lecture Notes in Comp. Sci.*, pages 116–123. Springer, 2008.

[LHP02] Herbert Leitold, Arno Hollosi, and Reinhard Posch. Security architecture of the austrian citizen card concept. In *ACSAC*, pages 391–402. IEEE Computer Society, 2002.

[LL84] Jennifer Lundelius and Nancy A. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204, 1984.

[LLA+02] Sang Seok Lim, Chul Lee, Chul Woo Ahn, Chang Gyu Lee, and Kyu Ho Park. An adaptive admission control mechanism for a cluster-based web server system. In *IPDPS*. IEEE Computer Society, 2002.

[Lu07] HongQian Karen Lu. Network smart card review and analysis. *Computer Networks*, 51(9):2234–2248, 2007.

[LY09] Chung-Horng Lung and Oliver W. W. Yang. Evaluation of an adaptive PI rate controller for congestion control in wireless ad-hoc/sensor networks. In *CSE (2)*, pages 597–602. IEEE Computer Society, 2009.

[MAQ99] Henri Massias, X. Serret Avila, and Jean-Jacques Quisquater. Timestamps: Main issues on their use and implementation. In *WETICE*, pages 178–183. IEEE Computer Society, 1999.

[Mil06a] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC Press, Inc., Boca Raton, FL, USA, 2006.

[Mil06b] David L. Mills. Network time protocol version 4 reference and implementation guide. Technical Report 06-6-1, University of Delaware, June 2006.

[MJ09a] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Peer-to-Peer Comp.*, pages 99–100. IEEE, 2009.

[MJ09b] Lorenz Müller and Marcel Jacomet. Security device for online transaction. Patent, 12 2009. US 7636854.

[MJCLR07] Lorenz Müller, Marcel Jacomet, Roger Cattin-Liebl, and Alain Rollier. Method to transmit a coded information and device therefore. Patent Application, 06 2007. US 2007/0133839 A1.

[MK08] Amit Mondal and Aleksandar Kuzmanovic. Removing exponential backoff from TCP. *Computer Communication Review*, 38(5):17–28, 2008.

[MKY+07] Shunsuke Mogaki, Masaru Kamada, Tatsuhiro Yonekura, Shusuke Okamoto, Yasuhiro Ohtaki, and Mamun Bin Ibne Reaz. Time-stamp service makes real-time gaming cheat-free. In Grenville J. Armitage, editor, *NETGAMES*, pages 135–138. ACM, 2007.

[MM87] R Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, June 1987.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2002.

[MMBK10] D. Mills, J. Martin, J. Burbank, and W. Kasch. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June 2010.

[MO83] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 295–305. ACM, 1983.

[MO85] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. *SIGPS Oper. Syst. Rev.*, 19(3):44–54, 1985.

[MOZ+09] Pieter J. Meulenhoff, Dennis R. Ostendorf, Miroslav Zivkovic, Hendrik B. Meeuwissen, and Bart M. M. Gijsen. Intelligent overload control for composite web services. In Luciano Baresi, Chi-Hung Chi, and Jun Suzuki, editors, *ICSOC/ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 34–49, 2009.

[MPR05] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-is-believing: Using camera phones for human-verifiable authentication. In *IEEE Symposium on Security and Privacy*, pages 110–124. IEEE Computer Society, 2005.

[Mül08a] Lorenz Müller. Authentication and transaction security in e-business. In Simone Fischer-Hübner, Penny Duquenoy, Albin Zuccato, and Leonardo Martucci, editors, *The Future of Identity in the Information Society*, volume 262 of *IFIP International Federation for Information Processing*, pages 175–197. Springer Boston, 2008.

[Mül08b] Thomas Müller. *Trusted Computing Systeme*. Xpert.press. Springer, 2008.

[MvO07] Mohammad Mannan and Paul C. van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography*, volume 4886 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2007.

[NM08] T. Nishikawa and S. Matsuoka. Time-stamping authority grid. In *CCGRID*, pages 98–105. IEEE Computer Society, 2008.

[OBDS04] Alina Oprea, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Securing a remote terminal application with a mobile trusted device. In *ACSAC*, pages 438–447. IEEE Computer Society, 2004.

[OCK10] Clemens Orthacker, Martin Centner, and Christian Kittl. Qualified mobile server signature. In Kai Rannenberg, Vijay Varadharajan, and Christian Weber, editors, *Security and Privacy âĂŞ Silver Linings in the Cloud*, volume 330 of *IFIP Advances in Information and Communication Technology*, pages 103–111. Springer Boston, 2010. 10.1007/978-3-642-15257-3_10.

[PA00] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

[Pos81] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[PPBG09] Jérémy Philippe, Noël De Palma, Fabienne Boyer, and Olivier Gruber. Self-adapting service level in java enterprise edition. In *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 143–162. Springer Berlin / Heidelberg, 2009.

[PS99] Adrian Perrig and Dawn Song. Hash visualization: a new technique to improve real-world security. In *In International Workshop on Cryptographic Techniques and E-Commerce*, pages 131–138, 1999.

[Ran00] Kai Rannenberg. Multilateral security a concept and examples for balanced security. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms*, pages 151–162, New York, NY, USA, 2000. ACM.

[Roß08] Heiko Roßnagel. *Mobile Qualifizierte Elektronische Signaturen*. PhD thesis, Goethe-Universität in Frankfurt am Main, 2008.

[RV10] Julien Ridoux and Darryl Veitch. Principles of robust timing over the internet. *ACM Queue, Communications of the ACM*, 53(5):54–61, May 2010.

[Sch05] Bruce Schneier. Two-factor authentication: too little, too late. *Commun. ACM*, 48(4):136, 2005.

[SKK07] Martin Szydlowski, Christopher Kruegel, and Engin Kirda. Secure input for web applications. In *ACSAC*, pages 375–384. IEEE Computer Society, 2007.

[SO03] Pauline Sourdille and Aidan O'Dwyer. A new modified smith predictor design. In *ISICT*, volume 49 of *ACM International Conference Proceeding Series*, pages 385–390. Trinity College Dublin, 2003.

[SRS00] Preeti Bhoj Srinivas, Srinivas Ramanathan, and Sharad Singhal. Web2K: Bringing QoS to web servers, 2000. `http://www.hpl.hp.com/techreports/2000/HPL-2000-61.pdf`.

[SS99] Bruce Schneier and Adam Shostack. Breaking up is hard to do: modeling security threats for smart cards. In *WOST'99: Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*, pages 19–19, Berkeley, CA, USA, 1999. USENIX Association.

[Tra02] Transaction Processing Performance Council. TPC Benchmark^{TM}W (Web Commerce), 2002. `http://www.tpc.org/tpcw/`.

[Tul06] D. Tulone. A scalable and intrusion-tolerant digital time-stamping system. In *Communications, 2006. ICC '06. IEEE Int. Conf. on*, volume 5, pages 2357 –2363, 11-15 2006.

[Uri01] Pascal Urien. Smarttp smart transfer protocol. Internet Draft, June 2001.

[Uri09] P. Urien. TLS-tandem: A smart card for WEB applications. In *6th IEEE Consumer Communications and Networking Conf. CCNC 2009.*, pages 1–2, January 2009.

[VG02a] Thiemo Voigt and Per Gunningberg. Adaptive resource-based web server admission control. In *ISCC*, pages 219–224. IEEE Computer Society, 2002.

[VG02b] Thiemo Voigt and Per Gunningberg. Handling multiple bottlenecks in web servers using adaptive inbound controls. In Georg Carle and Martina Zitterbart, editors, *Protocols for High-Speed Networks*, volume 2334 of *Lecture Notes in Computer Science*, pages 50–68. Springer, 2002.

[VRK09] Darryl Veitch, Julien Ridoux, and Satish Babu Korada. Robust synchronization of absolute and difference clocks over networks. *IEEE/ACM Transactions on Networking*, 17(2):417–430, April 2009.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

[YLE04] C. K. Yeo, B. S. Lee, and M. H. Er. A survey of application level multicast techniques. *Computer Communications*, 27(15):1547–1568, 2004.

[ZLM07] Huang Ziyuan, Zhen Lanlan, and Fei Minrui. Robust auto tune smith predictor controller design for plant with large delay. In Kang Li, Minrui Fei, George W. Irwin, and Shiwei Ma, editors, *LSMS (1)*, volume 4688 of *Lecture Notes in Computer Science*, pages 666–678. Springer, 2007.

[ZN42] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Transactions of the ASME*, 64:759–768, 1942.

List of Figures

Index

Publications

[1] G. Starnberger, L. Froihofer, and K. M. Goeschka, "Adaptive run-time performance optimization through scalable client request rate control," in *Proc. 2nd Joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW'11)*, ACM, March 2011.

[2] G. Starnberger, L. Froihofer, and K. M. Goeschka, "Using smart cards for tamper-proof timestamps on untrusted clients," in *ARES 2010, Fifth International Conference on Availability, Reliability and Security, 15-18 February 2010, Kraków, Poland*, pp. 96–103, IEEE Computer Society, 2010.

[3] G. Starnberger, L. Froihofer, and K. M. Goeschka, "Distributed timestamping with smart cards using efficient overlay routing," in *Fifth International Conference for Internet Technology and Secured Transactions (ICITST 2010)*, Nov. 2010.

[4] G. Starnberger, L. Froihofer, and K. M. Goeschka, "QR-TAN: Secure mobile transaction authentication," in *Proceedings of the The Forth International Conference on Availability, Reliability and Security, ARES 2009, March 16-19, 2009, Fukuoka, Japan*, pp. 578–583, IEEE Computer Society, 2009.

[5] G. Starnberger, L. Froihofer, and K. M. Goeschka, "A generic proxy for secure smart card-enabled web applications," in *Web Engineering, 10th International Conference, ICWE 2010, Vienna, Austria, July 5-9, 2010. Proceedings*, vol. 6189 of *Lecture Notes in Computer Science*, pp. 370–384, Springer, 2010.

[6] G. Starnberger, C. Kruegel, and E. Kirda, "Overbot: a botnet protocol based on kademlia," in *SecureComm '08: Proceedings of the 4th international conference on Security and privacy in communication networks*, (New York, NY, USA), ACM, 2008.

Curriculum vitæ

Günther Starnberger

Year of birth: 1981

Email: guenther@starnberger.name
Web: https://guenther.starnberger.name/

Address: Allerheiligengasse 5/2/1
1200 Wien

Education

2007–2011 Doctoral Study in Computer Science
Vienna University of Technology

2005–2007 Software Engineering & Internet Computing (Dipl.-Ing.)
Vienna University of Technology

2005–2007 Informatikmanagement (Mag.)
Vienna University of Technology

2006 Joint Study Student Exchange Program
City University of New York

2006 Erasmus Student Exchange Program
Universiteit van Amsterdam

2001–2005 Software & Information Engineering (Bakk.)
Vienna University of Technology

Work experience

2008–2011	Vienna University of Technology Distributed Systems Group Research Assistant
2005–2006	EUnet GmbH Software Development & System Administration
1999–2005	ATnet Dr. Franz Penz Software Development & System Administration

Languages

German (native)
English (excellent)