RC 21851 (98343) 5 October 2000 **Computer Science**

IBM Research Report

WebALPS: Using Trusted Co-Servers to Enhance Privacy and Security of Web Interactions

Sean W. Smith
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

WebALPS: Using Trusted Co-Servers to Enhance Privacy and Security of Web Interactions

Sean W. Smith[*]
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory, Hanover NH 03755-3510 USA
sws@cs.dartmouth.edu

October 5, 2000

Abstract

This report describes a project that aims to fill two gaps in recent security and privacy research. The first gap is *trust.* Too often, "security of Web transactions" reduces to "encryption of the channel"—and neglects to address what happens at the server on the other end. This oversight forces clients to trust the good intentions and competence of the server operator—but gives clients no basis for that trust. The second gap is *secure coprocessing.* Despite early academic research in the potential of this technology, and the subsequent industrial research that resulted high-assurance, programmable secure coprocessors as COTS products, many in the computer science community still regard "secure hardware" as a synonym for "cryptographic accelerator.' This oversight neglects the real potential of COTS secure coprocessing technology to establish trusted islands of computation in hostile environments (e.g., at servers with risk of insider attack).

The WebALPS project proposes to address both issues by using secure coprocessors to establish trusted third parties at Web servers. Having clients establish an SSL session *into* an application running inside the secure hardware at the Web server (instead of just using secure hardware to speed cryptography) provides a systematic way to enhance the security of a broad family of Web-based services—including *security against insider attack*—without requiring a substantial change to the currently deployed Web infrastructure.

1 Introduction

For the last seven years, this author has spent much time in two arenas:

- security and privacy issues for Web-based e-commerce and e-government services; and
- technology and applications of secure coprocessing.

In both arenas, despite significant technological advances, fundamental problems remain:

- despite strong encryption on the browser-server channel, Web users still have no assurance about what happens at the other end;
- despite early academic work in secure coprocessing [18, 19] and the more recent emergence of high-performance, high-assurance programmable platforms (as COTS products, even), the potential of secure coprocessing has not been effectively demonstrated to the computer science community at large.

[*]This report presents work the author began in the Secure Systems and Smart Card group at IBM T.J. Watson Research Center (from which the author is on leave), and hopes to continue at Dartmouth.

This report presents a research project that can address both these issues. We propose:

- augmenting Web servers with *trusted co-servers* comprised of high-assurance secure coprocessors, configured with a publically known guardian program;
- training Web users to establish their authenticated, encrypted channels with a trusted co-server, which then can act as a trusted third party participating in the browser-server interaction.

A successful set of working prototypes would address the core trust problem in the Web—since the combination of the authenticated channel with the secure coprocessor would give Web users assurance in the correctness and privacy of sensitive computation at the server site, *even if the server operator has motivation to subvert these properties.* What's more, this work would also provide a practical demonstration of the potential of secure coprocessing to resolve difficult trust problems—with real code, for real problems, with a feasible commercial and technological path to broader deployment.

We have been calling this project *WebALPS:* Web Applications with Lots of Privacy and Security. (This name emerged at lunch one day, partly in jest: since it connotes "little Switzerlands": little neutral areas distributed on the Web.)

Prior Work Prior work has:

- established how to build tamper-responding secure hardware;
- established how to use this tamper-response technology (and many other techniques) to build general-purpose secure coprocessors, that can be trusted to carry out their computation correctly despite attempts to physically attack or modify this computation;
- examined the use of secure hardware for some individual specialized applications; and
- examined the use of secure hardware as SSL cryptographic accelerators for Web servers.

The WebALPS project unifies and moves beyond all of these, by using secure coprocessors to establish trusted third parties at Web servers. These trusted co-servers can then bring many additional security and privacy properties to a wide range of Web applications. Basing this work on having clients establish an SSL session *into* an application running inside the secure hardware at the Web server (instead of just using secure hardware to speed cryptography) provides a systematic way to enhance the security of a broad family of Web-based services, without requiring a substantial change to the currently deployed Web infrastructure.

We stress that, although some prior work discusses "secure web servers," we believe that we are unique in examining web servers that are *secure against insider attack.*

This Report

- Section 2 presents some background on secure coprocessing and on SSL.
- Section 3 presents the Web angle: manifestations of a basic trust problem (Section 3.1), the proposed WebALPS architecture (Section 3.2), and how it can address this problem in its many manifestations (Section 3.3).
- Section 4 presents the research angle: the basic secure coprocessor thesis (Section 4.1), the obstacles to proving this thesis to the larger community (Section 4.2), and how the WebALPS project can overcome these obstacles (Section 4.3).
- Section 5 presents the strategy for where we go from here.

Acknowledgments The author wishes to thank many colleagues. First and foremost, Nao Itoi provided much helpful discussion and suggestions, especially regarding implementation strategies; Joan Dyer and Ron Perez also gave implementation advice. Elaine Palmer suggested the logo license enforcement application (Section 3.3.9); Jeff Kravitz suggested the download authenticity application (Section 3.3.11); Dave Chess suggested the offline antivirus application (Section 3.3.10). Peter Gutmann and Doug Schales answered SSL and Web questions; Adrian Perrig and Dawn Song helped dream up the acronym; and Charles Palmer and Dave Safford provided general support and encouragement.

2 Background

This section presents some background material on secure coprocessing (Section 2.1) and SSL (Section 2.2).

2.1 Secure Coprocessing

The technology of high-performance, high-assurance programmable secure coprocessors enables the WebALPS project. In this section, we quickly review the evolution of this technology. Section 2.1.1 presents the motivating problem; Section 2.1.2 discusses how secure coprocessors address this problem; and Section 2.1.3 reviews the current state-of-the-art platform.

2.1.1 Motivation

Distributed computation (and even centralized computation, with multiple parties) introduces a fundamental problem: distribution dissociates dependency from control. Consider this basic scenario:

- Alice and Bob participate in some computational activity.
- Alice's interests $\mathcal{I}$ depends on some correctness and/or privacy properties $\mathcal{P}$ of some computation X at a computer that Bob controls.
- Consequently, Alice must depend on Bob to preserve and protect her interests.
- However, Bob may have no motivation to do this; and, in fact, Bob's interests may conflict with Alice's, and motivate him to actively subvert Alice's computation.

Figure 1 illustrates this basic scenario.

Basic Example This scenario characterizes many real-world scenarios. For just one example, consider a bank that uses a host machine for their signing keys. In this example:

- Alice is the CEO of the bank.
- Bob is the least-trusted employee.
- The computation X is the storage, generation, and application of these keys.
- Alice's interests $\mathcal{I}$ include:
  - her company's reputation,
  - and the assets protected by these keys.
- Preservation of $\mathcal{I}$ depend on properties $\mathcal{P}$ such as the secrecy, unpredictability, and correct application of these keys.
- However, any employee with access to the host (which, in most networked business environments, is any employee with access to any networked machine, and the inclination to learn some easily-acquired penetration skills) can access the host machine and subvert these properties.

Consequently, Alike incurs a dependency on Bob that she would rather not have.

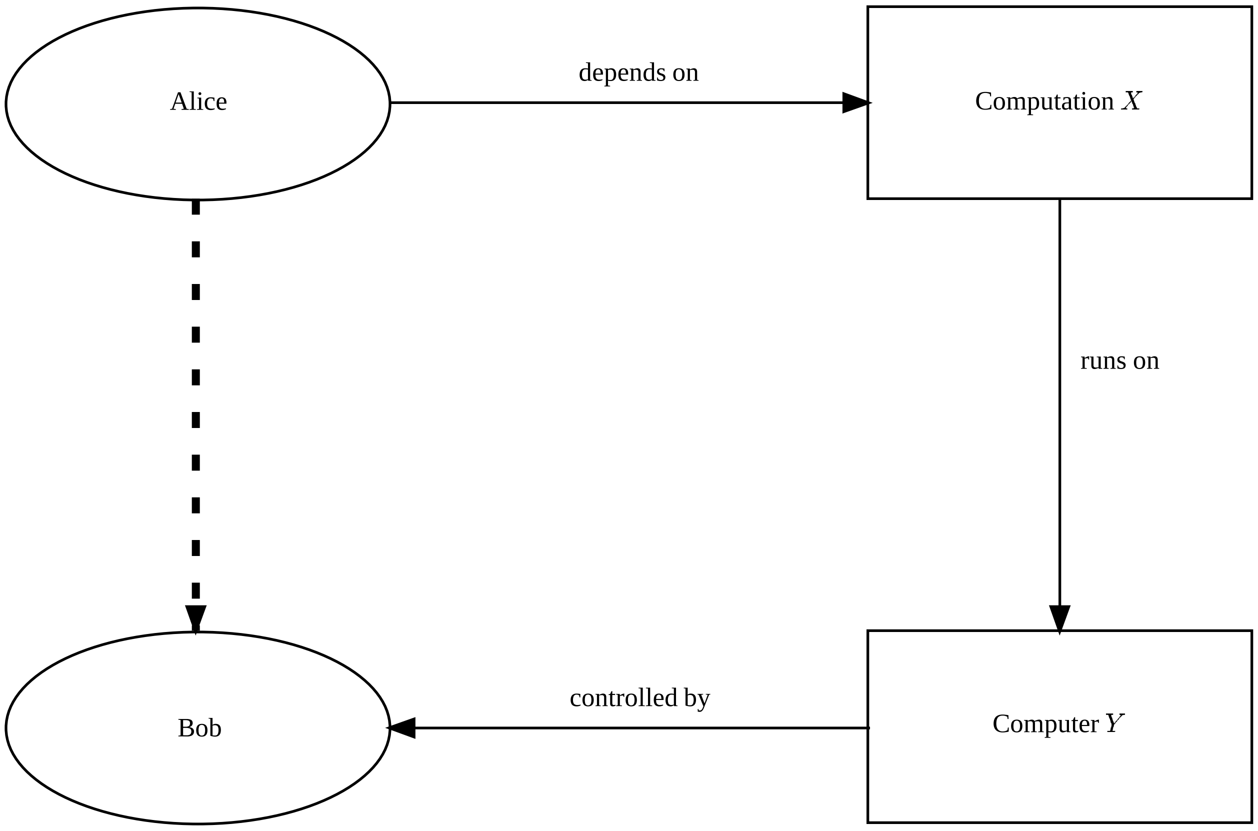

Figure 1 The basic motivating scenario: in distributed computation, one party (here, Alice) can incur a dependency on a party who may have different interests (here, Bob), because computation critical to Alice may reside on a machine that Bob controls.

More Advanced Examples In the above basic example, Alike incurs a dependency on Bob that she would rather not have. However, it is enlightening to consider other dependency scenarios:

- **Mutual Dependence.** Suppose Alice and Bob are users in a decentralized e-cash system [18], where cash is a value in a register in a wallet, and is exchanged by a protocol between the wallets. All parties in this distributed e-cash system must trust all other parties; in a sense, the least-trusted user has the ability and the motivation to subvert the entire system.
- **Reluctant Dependees.** Alice may incur a dependency on Bob that *Bob* would rather not have. For example, consider a commerce application where Bob is a service provider.
  - An ability to assert "customers can trust my service because they do *not* have to trust me" can be an effective marketing tool.
  - An inability to fraudulently manipulate the service might preserve Bob from various litigation and legal hassles.
- **Self Dependence.** It can also be enlightening to consider scenarios where Alice would benefit from the inability to subvert computation that occurs at her own site—for example, as a defense against insider attack.

2.1.2 Secure Coprocessing as a Solution

The technology of *secure coprocessors* has long been proposed as a foundation to address these problems. Quickly defined, a secure coprocessor is a general-purpose computer (possibly with cryptographic support) that is secure against all foreseeable physical and logical attacks. Secure coprocessors can augment the security of an otherwise untrusted host computer, because the relevant parties have the potential to trust that the computation and data storage occurring in the coprocessor have been unmolested by adversaries with direct physical access to the machine.

Programmable, trusted secure coprocessors could enable systematic solutions to problems such as Figure 1:

- If X occurred in a secure coprocessor at Bob's machine, and
- Alice was able to authenticate that X was occurring there, beyond Bob's control,
- and Bob's ability to manipulate his host and its network connections could not subvert $\mathcal{P}$,
- then Alice can trust that the important properties $\mathcal{P}$ still hold of X, despite Bob's potential attacks.

Figure 2 sketches this revised scenario.

Some individual examples of these trust problems may be solvable—perhaps at a significant performance cost–via special cryptographic protocols (such as *secure multiparty computation*). However, we still believe that only secure coprocessing can enable a systematic approach to high-performance solutions.

2.1.3 The State of the Art

The IBM 4758, developed in the Secure Systems group at IBM Watson, represents the state of the art in high-assurance high-performance, programmable secure coprocessors.

To a large extent, secure coprocessing research started at IBM Watson [7, 14, 15, 17], continued to other sites influenced by Watson [18, 8], and returning to Watson again [1, 4, 11, 13]). A significant amount of recent work at Watson [1, 4, 11, 13] has centered on defining and building such a programmable secure coprocessor as a product, and on validating it against the highest possible external standards [3, 6] in order to establish its trustworthiness to third parties [5, 9, 10].

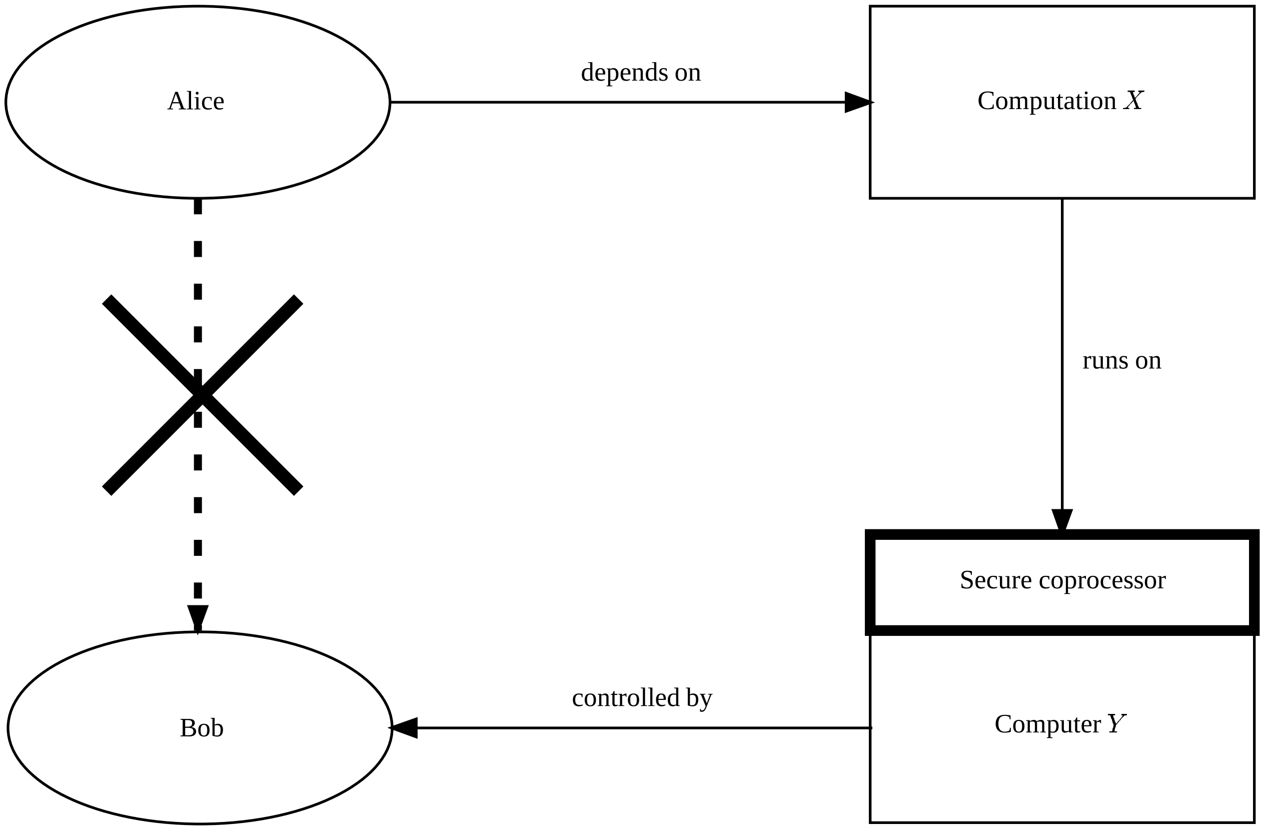

Figure 2 With secure coprocessing, Alice can trust the critical correctness properties of X, even if Bob may have motivation to subvert them. Here, by moving the computation on which Alice depends from Bob's machine to a secure coprocessor added to Bob's machine, Alice no longer incurs a dependency on Bob.

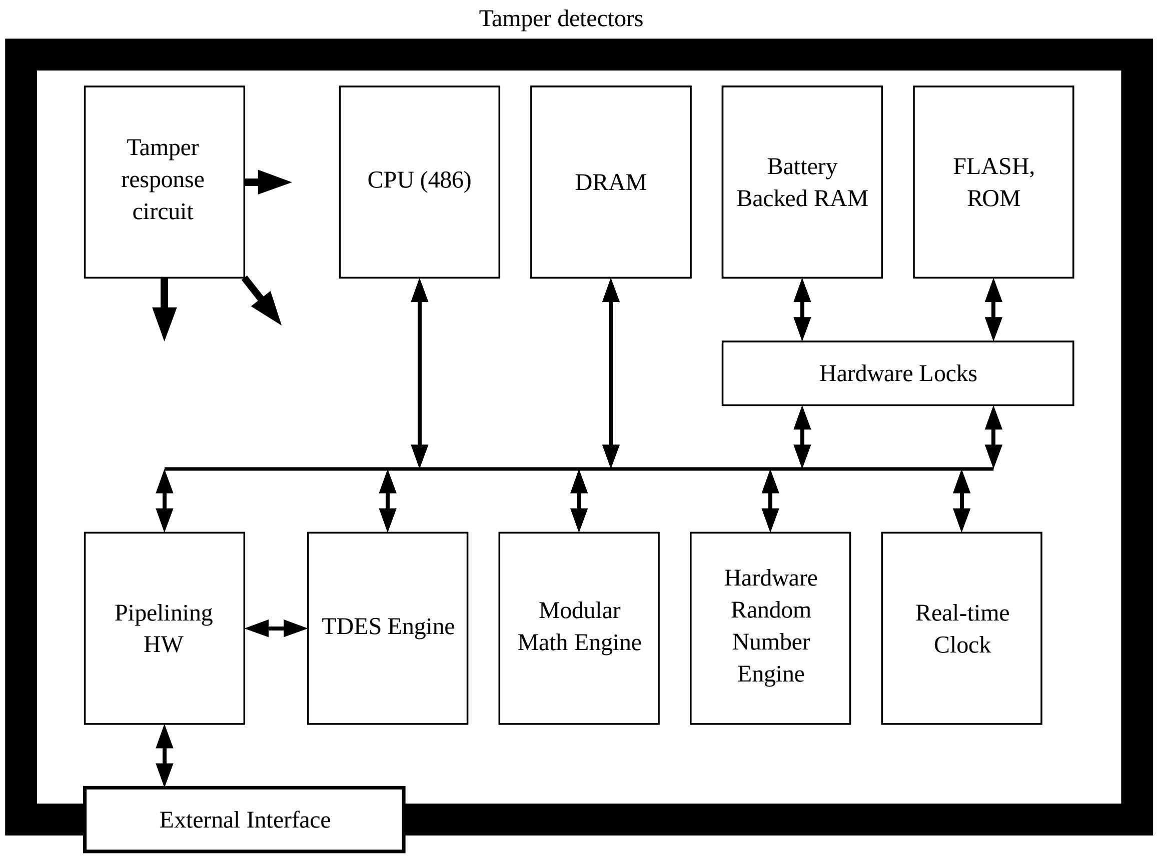

Figure 3 Basic hardware of the IBM 4758-2 secure coprocessor

Hardware Figure 3 sketches the hardware of the 4758. The device provides general-purpose computing environment for applications, with hardware support for cryptographic applications. However, the device also provides crucial security features:

- Continuously active tamper-detection circuitry monitors tamper detectors and, in case of physical attack, destroys sensitive secrets in secure memory before an adversary can access them.
- Hardware locks protect crucial code and secrets from possibly malicious or faulty application code, preserving the ability of each device to properly authenticate its configuration, and preventing a device with a rogue application from impersonating other devices and applications.

Software The IBM 4758 features a software architecture that permits application developers to install and update their applications into 4758 devices at customer sites, in such a way that protects the privacy and security interests of the developers, the customers, and IBM; see our security design paper [13] for more information here.

The Model 2 family of the IBM 4758 includes full support for *outbound authentication*, which enables on-board applications to, at run-time, authenticate their identity (and status as applications running on untampered hardware) to remote entities.

Figure 4 sketches the software configuration architecture:

- The coprocessor is shipped with low-level firmware, and knowledge of the vendor's public key.

- The coprocessor vendor gives an application developer:
  - a unique identifier;
  - a signed command telling coprocessors to with no application to recognize that developer as their application owner; and
  - a signed command telling coprocessors that that application developer has a specified public key.
- The application developer then signs his application code with his private key, and gives this signed code, along with the vendor-provided commands, to the end user.
- The end user provides these items to the security configuration software within the secure coprocessor. This software validates the commands against the vendor's public key and other parameters in the parameters store. If things validate, the security configuration software takes these steps:
  - it updates the parameter store to record that the application developer now owns the application space within this device, and records the developer's idea and public key;
  - it installs the application as the device's application software;
  - it generates a keypair for this application installation on this device;
  - it uses the device's own keypair to certify that this new keypair belongs to that application, for that owner, in that device;
  - and it leaves this application keypair in a place where the application software can access it at run-time.

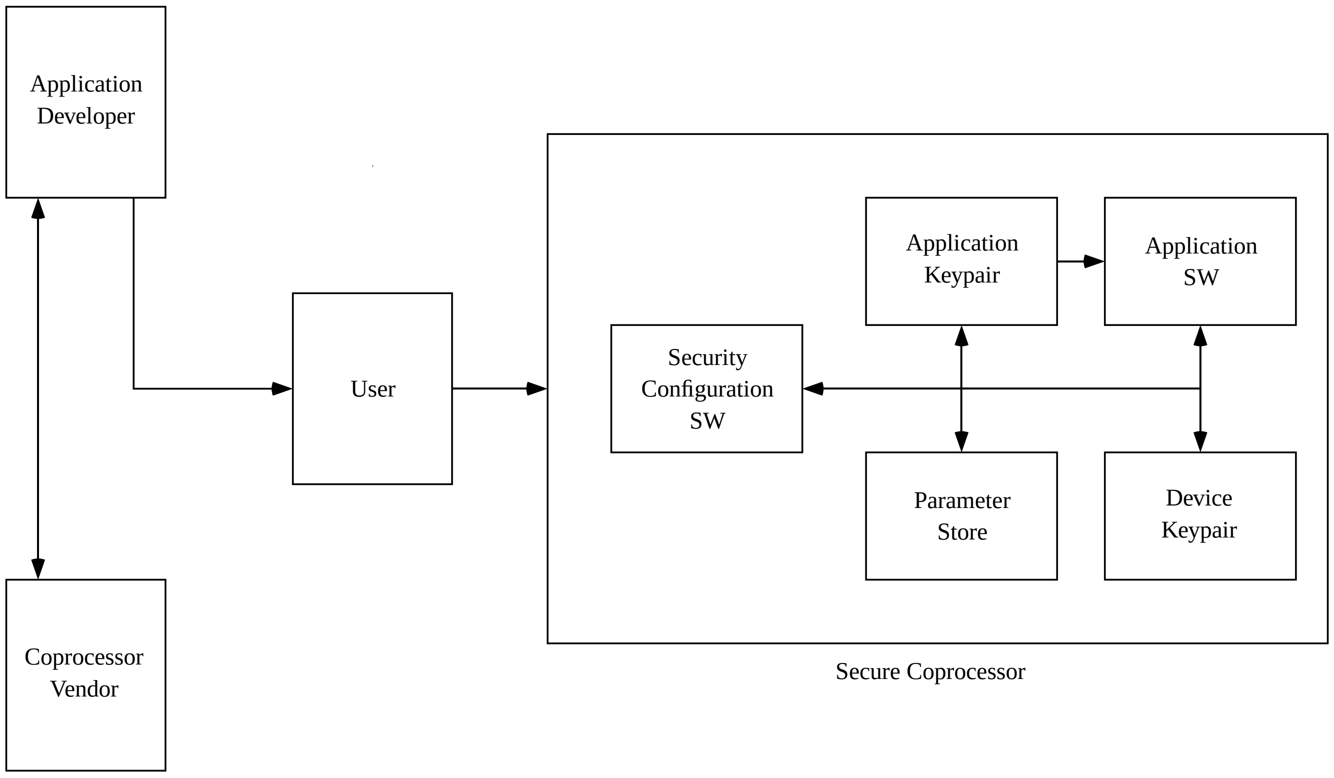

Figure 4 This figure sketches the software configuration architecture. Coprocessors start out life with their own certified device keypair, and knowledge of their vendor's public key. The coprocessor vendor signs the application developer's public key, and signs a command telling the security configuration software in a virgin card to accept application configuration commands from that developer. As an atomic part of the initial application installation and each subsequent configuration change, the security management software generates a new application-level keypair and certifies it to belong to this device, in this configuration. This application-level keypair enables outbound authentication: the ability of the the device to prove, to the outside world, that it's "the real thing, doing the right thing."

2.2 SSL

Readers not familiar with SSL may benefit from a quick introduction.

The World Wide Web was invented in 1994 by lazy physicists who hoped that a hypertext medium would help them avoid going to the library to look up references.

As the popularity of the Web—and the recognition of its potential for applications with real security issues—spread, many proposals and ideas surfaced to add security to the basic `http` protocol. At one point, three primary contenders emerged:

- *Shen,* from CERN (who originated the Web);
- *Secure HTTP,* from a consortium including NCSA (who produced Mosaic, the breakthrough browser); and
- *Secure Socket Layer (SSL),* from Netscape, a spinoff of NCSA.

Primarily because Netscape's SSL protocol was the first to be widely deployed, SSL became the de facto standard for securing Web transactions. The somewhat accidental nature of its emergence (like the Web's) has made for a somewhat interesting situation.

- The protocol, as typically deployed, unfortunately omits some security features that would have been very convenient—such as practical authentication of clients, and digital signatures of web content and form responses.
- The definition of the protocol, as typically deployed, is not controlled by a small design team but by the ebb and flow of features various browsers and servers support—or do not support.
- A precise and clear definition of the protocol is very hard to obtain. We have found that many questions are best answered not by consulting the existing documentation but rather by consulting some experienced implementers.

As practiced, SSL permits the client to establish a shared symmetric key with a specific authenticated server:

- The server has a private-public keypair, and a certificate from some CA attesting to something vague about the entity owning this public key.
- The client browser has some notion of which CA root keys it recognizes as valid.
- When a client opens an SSL connection, it verifies that the certificate from the server is correctly signed by a CA root that the client's browser currently recognizes as legitimate.
- The client and server then carry out a key generation/exchange protocol that ensures that the client, and a party which knows the private key matching the server's public key, share a symmetric key—that is (theoretically) shared by no one else, not even an adversary observing the messages between the client and server.

The remainder of the SSL session is then encrypted with this session key. Encryption with a key obtained this way provides several properties:

- The client can trust the privacy of data going to the server, and coming back.
- Both parties can trust that an adversary cannot alter or manipulate data in either direction without detection (since SSL provides integrity checking and sequence numbering).
- The client can trust the authenticity of the server (since the server entity must know the private key matching the public key in the certificate).
- The server can trust that, throughout the session, the entity claiming to be the client is the same entity that started the session (or, at worst, a colluding partner).

The protocol allows for some variations:

- The definition permits RSA-based key exchange or Diffie-Hellman key exchange; however, the main CAs do not issue Diffie-Hellman certificates, and it is conjectured that the main browsers probably do not implement it correctly.
- The definition permits a choice (through negotiation) of symmetric key and integrity checking protocols. (For performance, we should encourage WebALPS to use things like DES, TDES, and SHA-1, which the IBM 4758-2 supports in native hardware—if the main browsers also support these protocols.)
- The definition also permits the server to authenticate the client; however, this step is not supported in practice.

3 Adding Privacy and Security to Web-Based Services

This section presents how WebALPS can systematically add privacy and security to Web services. Section 3.1 discusses a fundamental trust problem in the current Web infrastructure; Section 3.2 presents the WebALPS architecture; and Section 3.3 discusses how this architecture can resolve this problem.

3.1 Fundamental Trust Problem

The World Wide Web is the grounds where, on a broad scale, our society's business, government, and personal services are migrating and growing. As a basic model, a large population of clients with browsers obtain services from a smaller population of service providers operating Web servers.

However, for each critical service that takes root in the Web (and arguably for many purely recreational services as well), the financial and personal interests of the clients forces them to trust the integrity and privacy of the computation and data storage at the service providers. But with the current state of deployed technology (i.e., SSL), all the client can be sure of is what the CA claims was the identity of the entity who originally possessed the public key in that server's certificate.

At best, this identity establishes good intentions—if the alleged service provider has a pre-existing reputation that makes this hypothesis plausible.

On the other hand:

- A service provider with an unknown reputation might be downright malicious.
- Any service provider may have good intentions, but may be careless with general site security.
- The entity with which the client is currently interacting may not even be this original service provider, but rather an imposter who has learned the private key.

The threat that arises from this uncertainty is amplified by the Web's distribution of computation from server to client: via Java and Javascript, and also via more subtly executable content, such as Word documents infected with Macro viruses.

Furthermore, many interactions involve more parties than just the client and server—and these additional parties are *also* forced to trust the server integrity.

This situation leads to a *fundamental trust problem:*

Participants in distributed Web services are forced to trust server integrity, but have no basis for this trust.

This problem threatens a wide variety of Web applications. In Section 3.1.1 through Section 3.1.12, we consider a few.

3.1.1 Problem: Nonrepudiable Authentication of Clients

The current Web infrastructure prevents a server from being able to prove anything to a third party about the *identity* of an alleged client.

Without a public-key infrastructure for citizens, clients are forced to use human-usable authenticators, such as userids and passwords. However, in the current infrastructure, these authenticators are exposed to the server of unknown integrity. As a consequence of this exposure, an adversary who compromises the server (or a malicious server operator) can impersonate this user at that site, and at any other site where the user has used these authenticators. This exposure also prevents legitimate server operators from being able to argue that it really was a particular client who opened a particular a session.

3.1.2 Problem: Nonrepudiation of Client Activity

The current Web infrastructure prevents a server from being able to prove anything to a third party about the *activity* of an alleged client.

For example, how can an insurance company taking an application from Alice over the Web turn around and prove that Alice really answered that question that way?

3.1.3 Problem: Nonrepudiation of Server Activity

The current Web infrastructure prevents a server from being able to prove anything to a third party about the activity of that server in an interaction.

For example, consider trying to prove something about the *questions* that generated the answers a client provided.

- Case law already exists that permits, in paper interactions, a client to alter a waiver before signing it—and if the service provider accepts the form without noticing the alteration, he is bound by it.
- Colleagues of the author have reported difficulty in U.S. Government security clearance processes, because an answer to a question five years ago was not consistent with the current revision of that question.

Do the responses to Web forms typically include the questions (or a hash of them)? If the clients are not even signing responses, how can we expect them to sign questions too?

3.1.4 Problem: Credit Card Transaction Security

The current Web infrastructure provides secure transmission of a client's information to the server—but what happens there is anyone's guess.

For example, consider the credit-card information and transaction amount a client sends when he wishes to purchase something. An adversary who compromises the server (or a malicious server operator) can use this data to carry out lots of mischief:

- He can increase the amount of the transaction.
- He can retain the amount but repeat the transaction many times.
- He can use the credit card information to forge additional transactions.

This situation may significantly reduce the potential market for new e-merchants without a pre-established reputation. ("Ribo's Books has a cheaper price than well-known Amazon, but how do I know that unknown Ribo won't steal or accidentally divulge my credit card info?")

3.1.5 Problem: Taxes on E-Commerce Activity

The current Web infrastructure provides no acceptable means to balance the legitimate interests of a third party to accurately learn certain information about individual or collective Web interactions, with the privacy interests of the other participants.

For example, consider the problem government's tax collection service trying to learn how much sales tax an e-merchant owes them for last month.

- Reporting *all* transactions to the government would be unacceptable to the merchant and customer for privacy concerns.
- Reporting only a total amount owed would be unacceptable to the government, since the figure would be unverifiable, and the merchant reporting this unverifiable figure would be motivated to understate it.

3.1.6 Problem: Re-Selling of Intellectual Property

The current Web infrastructure provides no acceptable means for a third party who participates in an interaction indirectly, by licensing proprietary information to the server, to protect their legitimate interests.

For example, a publisher who owns a large copyrighted image database might wish to make this available to a university library—but might worry that compromise of the university server will compromise the database.

3.1.7 Problem: Privacy of Sensitive Web Activity

The current Web infrastructure provides no means for a server operator to plausibly deny that they (or an adversary who has compromised their machine) is not monitoring all client interactions for nefarious purposes.

We present some examples from several domains:

- **Commerce.** Suppose a corporation provides a patent server as a public service. How can its competitors know for sure that the server operator is not data-mining their queries, in order to learn details of their proprietary research and development efforts?
- **Defense.** A unit preparing a special operation may request many maps for the particular region of interest. What prevents the operator of the map server from observing this sequence of requests—all from the same unit and focused on the same region—and drawing a conclusion about the existence and target of the planned operation?
- **Social.** Consider people who wish to obtain sensitive literature—about health topics, for example, or about currently unfashionable politics. What prevents the server operator from learning of their activity, and acting in a manner (such as informing employers or health insurers) in way that would unjustly compromise their privacy?

3.1.8 Problem: Correctness of Web Activity

The current Web infrastructure provides no means for a server operator to establish that they (or an adversary who has compromised their machine) has not otherwise altered or corrupted important correctness properties of the service.

Suppose an auction server provides a bulletin board service where customers can post "timestamped, anonymous, confidential" comments about participants and interactions. How can customers know that the anonymous posts came from bona fide customers, and that the timestamps are correct?

3.1.9 Problem: Enforcement of Logo Licenses

The current Web infrastructure provides no effective means for a party to ensure that logos or endorsements appear only on the appropriate server pages.

For example, IBM could establish an "IBM-inspected" logo to endorse servers who have withstood penetration testing by IBM Global Services. However, any client who visits these pages can capture the logo, and put it on any page, whether or not that site has withstood the testing.

3.1.10 Problem: Safety of Downloadable Content

The current Web infrastructure provides no means for the client to ensure that executable content downloaded from a server is indeed safe.

With the current state of consumer platforms, safety depends on the client themselves actually running the latest anti-virus software. Since most consumers do not do this, this leaves them at risk. Moving the virus-checking computation (and the concomitant problem of maintaining the latest updates of virus signatures) to the server is more efficient—but how can clients know the server really carried this out?

3.1.11 Problem: Authenticity of Downloadable Content

The current web infrastructure provides no easy means for the client to authenticate the origin of downloadable content.

Posters of content can provide digital signatures, but then the client needs to explicitly obtain and verify the trust chain on each item. Moving this verification computation (and the concomitant problem of maintaining the latest certificate revocation lists) to the server is more efficient—but how can clients know the server really carried this out?

3.1.12 Problem: Integrity of Server Machine

The current Web infrastructure provides no means for the client to verify the integrity and site security of server machines.

For example, some servers may run on machines whose administrators who run hardened operating systems and/or engage in other good security practices, such as regular runs of a network security analyzer, or enhanced OS boot via secure hardware (e.g., [18]).

But any site can *claim* to do this. How can a client know?

3.2 The WebALPS Architecture

To solve these problems, we need a systematic way for clients to trust the computation and data storage that occurs at a Web service provider of unknown credibility.

What we propose is:

- adding a secure coprocessor to the existing service provider infrastructure, as a *trusted co-server*;
- installing one of a well-defined, reasonably small set of guardian programs at the trusted co-server;
- training clients to use SSL to open an authenticated connection to the WebALPS guardian program at the co-server, not to the server directly;
- changing the client-server interaction to begin as client-co-server interaction, and then bringing in the server as appropriate.

3.2.1 Properties

The WebALPS guardian program on the trusted co-server becomes a trusted "neutral" participant in the client-server interaction. By proper design of guardian program and server interaction, we can address the fundamental web security problem by raising the trust level of the computation and data storage at the server.

For example:

- The WebALPS guardian can witness to authenticity of certain data coming back to the client. This data can include assertions from the trusted guardian about the server content and configuration.
- The WebALPS guardian can provide privacy of data going back to the server, by keeping it encrypted between the client and the guardian, and then re-encrypting it before inserting it into the server.
- The user can trust the integrity of the computation occurring at the WebALPS guardian—even if the server operator might be motivated to subvert it.
- For computation relevant to third parties who may also have an interest in the client-server interaction, WebALPS provides a haven that all parties can trust to balance their interests.

Naively, one could also address such trust problems by putting the entire back-end service operation inside a coprocessor—-but this does not constitute an effective solution. The WebALPS approach avoids the problems of this naive straightforward approach:

- **Size.** The computational power, performance, and storage requirements for many Web servers probably exceed that provided by the state-of-the-art IBM 4758 coprocessor (Section 2.1).

  With WebALPS, rather than putting the entire server backend into a coprocessor, we're only putting a small guardian.
- **Legacy.** Even if we could fit entire operations inside a coprocessor, such solutions would require substantial changes to existing e-merchants and service providers, who already have existing back-end systems.

  With WebALPS, rather than replacing the legacy server backend, we're merely adding an additional component.
- **Authentication.** In order for the client to trust the server operations occurring in a secure coprocessor, the client has to be able to authenticate that the operation really is occurring in that type of system. Otherwise, we're back to "good intentions" again.

  With WebALPS, the proposed location of the guardian lets it easily use the existing server-side authentication features of the current infrastructure.

- **Usability of Authentication.** Suppose we could fit the service software inside a coprocessor and could alter the client infrastructure so that clients could authenticate it really was that software. Each service provider potentially has their own software configuration. How could clients make any practical trust judgments about the server based solely on the identity of that software?

  With WebALPS, by limiting ourselves to a set of well-defined guardian programs, users do not need to make a new trust decision for each server configuration—but, rather, for each guardian program.

(Authentication of the human at the client end is also an issue, but harder to address. E-Merchants seem to be solving this just fine by adopting the legacy credit card system. Providers of government services, such as the Social Security Administration's PEBES experiments, seem to be faring more poorly.)

3.2.2 Architecture

We assume the existence of:

- some well-defined class $\mathcal{A}$ of WebALPS guardian types, each denoted by a human-understandable name;
- a trusted WebALPS Certificate Authority; and
- some mechanism in client browsers to clearly indicate:
  - which portion of the browser window exclusively renders material from the other entity in this SSL session;
  - when an SSL session has been established from a certificate signed the WebALPS CA;
  - the identity of the member of $\mathcal{A}$ to which this certificate belongs.

Certification Figure 5 shows an example process for a server operator to use a high-end secure coprocessor platform (such as the 4758) to install and certify a trusted co-server.

The server operator obtains a secure coprocessor platform, and uses its software configuration architecture to install co-server application software $A \in \mathcal{A}$ from a co-server software vendor into this device. The co-server application then generates a new keypair *KP*. The server operator uses the co-server application's ability to authenticate itself with the co-server keypair to prove to the satisfaction of the WebALPS certificate authority that the new keypair *KP* belongs to an installation of A securely running on an untampered secure coprocessor platform at this server. The WebALPS Certificate Authority then issues an SSL-compatible certificate attesting to the public key of *KP* and the entity (WebALPS application A, running inside this secure coprocessor at this server) to which it belongs. The WebALPS application stores this certificate, and is then ready to participate as a trusted co-server to server operator's web application on his web server).

SSL Figure 6 shows an example establishment of a secure SSL session between client and trusted co-server.

A remote client using her Web browser initiates an SSL session with the co-server application within the secure coprocessor at the web site maintained by this server operator. Because the client's web browser indicates that the co-server application suitably demonstrates knowledge of the private key matching the public key in this application's SSL-certified keypair, the client can reasonably conclude that:

- server-client communications within this SSL session originated within the trusted co-server, and that
- client-server communications terminate inside the trusted co-server.

Furthermore, these conclusions hold even if the server operator may be motivated to maliciously alter or spy on these communications.

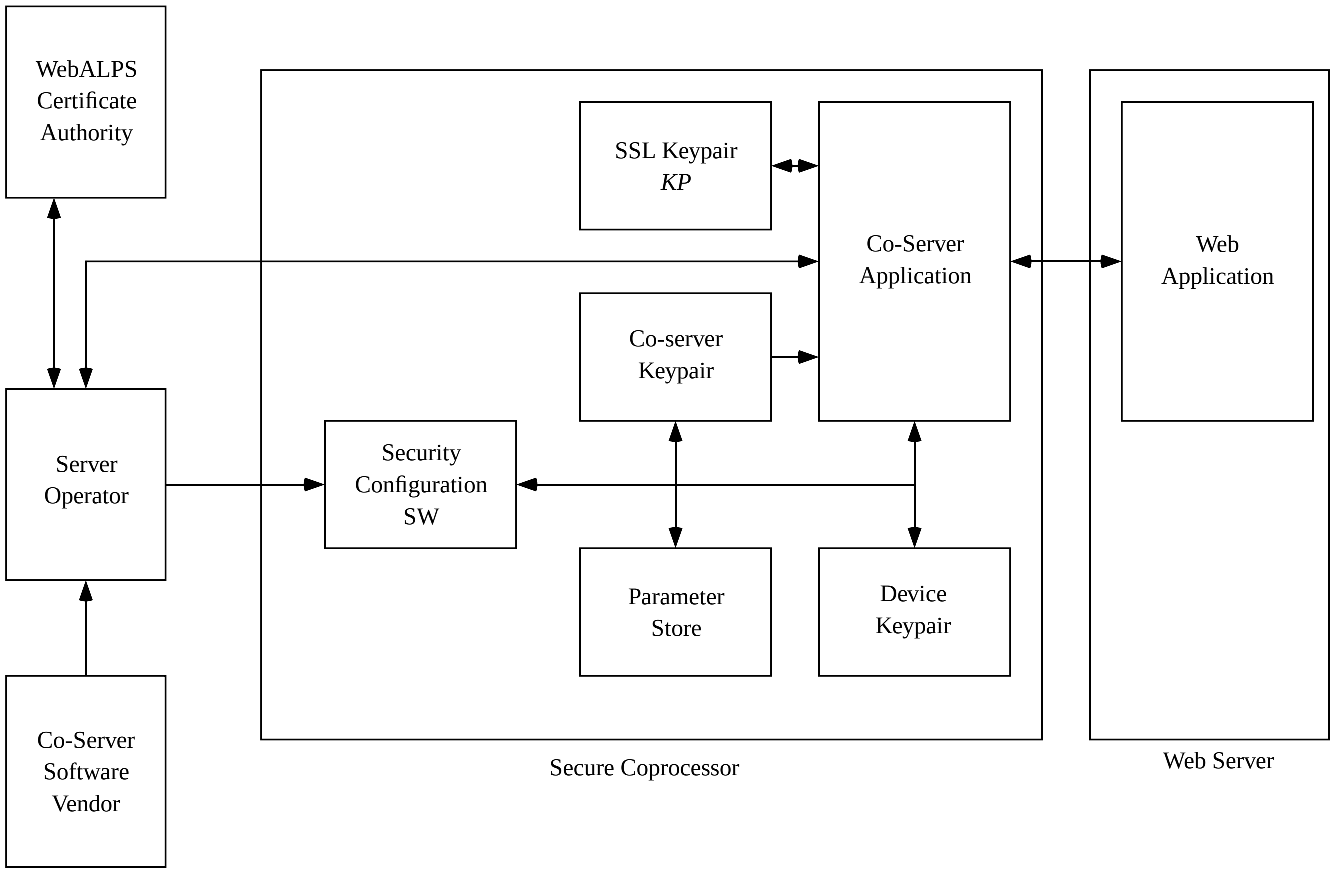

Figure 5 This figure sketches an example WebALPS installation and certification scenario. The server operator obtains a coprocessor from the coprocessor vendor, and the desired WebALPS guardian program (with all the necessary signatures) from the software vendor. The coprocessor's security configuration software installs the guardian, which then generates an SSL keypair *KP* and uses the coprocessor's outbound authentication to prove, to the CA, that this genuine guardian know the private key in this pair. The CA then issues an SSL-compatible certificate for the public key in *KP*.

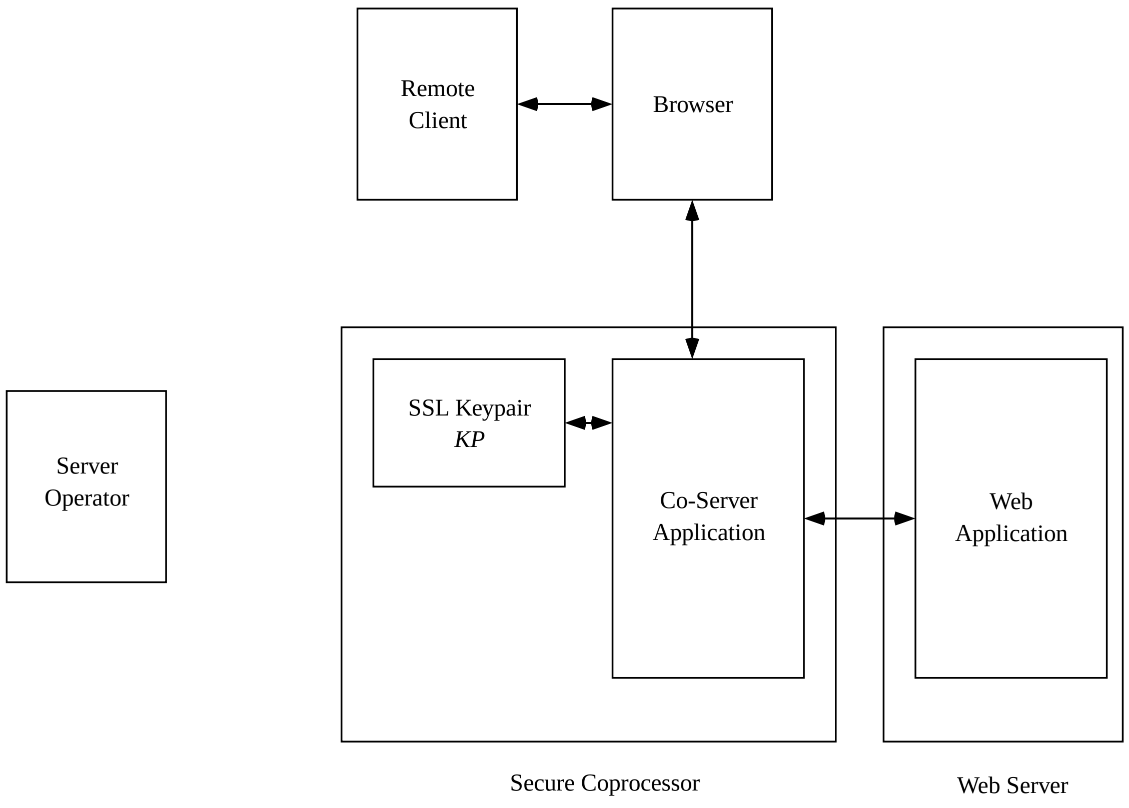

Figure 6 This figures sketchs example establishment of secure SSL session between client and trusted co-server. The remote client initiates an SSL session with the trusted co-server: the WebALPS program executing inside the coprocessor. Since the private key in *KP* is safely confined inside the coprocessor, the server operator cannot know the session key and thus cannot eavesdrop or forge communications between the browser and the co-server. The co-server can then work with the host Web server to provide enhances service to the remote client.

3.3 Solving the Fundamental Problem

Integrating WebALPS into the Web infrastructure solves the basic problem of how clients can establish trust in the computation and data storage at servers of otherwise unknown credibility.

We enumerate the (informal, for now) trust assertions.

- Because of the security architecture of the tamper-responding secure coprocessors, the client Alice can trust that the only entities who can authenticate themselves as an instance of a particular type of WebALPS guardian are in fact bona fide WebALPS guardians of that type.
- Because she trusts the WebALPS Certificate Authority to do its job, the client Alice can trust that the only entity possessing a private key matching the public key in a WebALPS certificate for a member of $\mathcal{A}$ is a bona fide WebALPS guardian of that type. (Essentially, to simplify the authentication process, Alice delegates the details of authenticating a tamper-responding coprocessor in a particular software configuration to the CA.)
- Because of the nature of the SSL protocol, the client Alice can trust that, when she opens an SSL session to some entity, she has established an encrypted channel between her browser and an entity that possesses that private key.
- Because her browser tells her unambiguously that this entity had an SSL certificate signed by the WebALPS CA, the client Alice can trust that this entity is a WebALPS guardian of that type.
- Because a well-defined portion of her browser window exclusively renders material from that SSL entity, Alice can trust that the data she receives and sends via this window passes through this encrypted channel to a bona fide WebALPS guardian of that type.

By way of example, WebALPS can solve the trust problems in each of the example applications from Section 3.1. In each case, we could design a particular type of WebALPS guardian; clients would check that it was that type of guardian before proceeding. In Section 3.3.1 through Section 3.3.12, we now consider these solutions.

3.3.1 Solution: Nonrepudiable Authentication of Clients

To solve the trust problem of Section 3.1.1, the WebALPS guardian at the server can trap the password, authenticate the client, then issue a signed receipt for the server that client properly authenticated for that session.

The client can trust that the password safely remains inside the coprocessor, even if the server operator might be motivated to retrieve it. The server operator, additionally, has a signed statement from a mutually trusted third party (the WebALPS guardian) that the alleged client really did authenticate correctly.

3.3.2 Solution: Nonrepudiation of Client Activity

To solve the trust problem of Section 3.1.2, the WebALPS guardian of Section 3.3.1 above can also issue a signed receipt for the entire transaction. "Alice not only authenticated correctly, but she issued a request of type X with parameters Y."

3.3.3 Solution: Nonrepudiation of Server Activity

To solve the trust problem of Section 3.1.3, the WebALPS guardian of Section 3.3.2 above can include, in its signed receipt for the transaction, the prompts and responses the server provided.

3.3.4 Solution: Credit Card Transaction Security

To solve the trust problem of Section 3.1.4, the WebALPS guardian at the server can trap the credit card and transaction information, and then inject it directly into the acquirer's system. The CCN data never appears in plaintext at the server site; the server operator or a penetrator has no opportunity to inflate the transaction amount; and (unlike SET) the client need not change the way she operates.

3.3.5 Solution: Taxes on E-Commerce Activity

To solve the trust problem of Section 3.1.5, the WebALPS guardian at the server can monitor the total tax owed by that merchant for the transactions that went through it (e.g., because of some other co-server application there), and report that authenticated total back to the government revenue agency. The agency can trust that the reported amount is correct; the merchant and customers can trust that the agency learns only what it is supposed to—and, in particular, is *not* learning details of transactions or identities of customers.

3.3.6 Solution: Re-Selling of Intellectual Property

To solve the trust problem of Section 3.1.6, the WebALPS guardian at the server would be set up to receive a session key and licensing rules from the owner of the IP. The owner would provide the IP in ciphertext to the server; the guardian would decrypt the particular items being used, and ensure that whatever licensing/royalty/watermarking requirements were being enforced.

3.3.7 Solution: Privacy of Sensitive Web Activity

To solve the trust problem of Section 3.1.7, the WebALPS guardian would be set up with the appropriate *private information retrieval* software (e.g., [12]) that makes use of (encrypted) storage in the server's file system. The client, through the SSL-protected channel, makes her request to the WebALPS guardian, which then retrieves the record, re-encrypts it, and returns it to the client. If we assume that cryptography works, the server operator learns nothing except the fact that a query has been made.

3.3.8 Solution: Correctness of Web Activity

To solve the trust problem of Section 3.1.8, we move the computation critical to the appropriate correctness properties from the server into the WebALPS guardian—whose application program would need to advertise that it was performing these computations.

3.3.9 Solution: Enforcement of Logo Licenses

To solve the trust problem of Section 3.1.9, the WebALPS guardian would be set up to provide the logo information, when appropriate. Logos that do not appear in the portion of the browser window from an authenticated guardian-to-client channel are not legitimate.

3.3.10 Solution: Safety of Downloadable Content

To solve the trust problem of Section 3.1.10, the WebALPS guardian could run the anti-virus software with the latest signatures:

- either dynamically, as the guardian was feeding data back to the client;
- or offline (but then, when the guardian was feeding data back to the client, it would verify that it had indeed scanned this data earlier).

Clients could then trust that content downloaded via this SSL-authenticated channel from the trusted co-server has been scanned.

3.3.11 Solution: Authenticity of Downloadable Content

To solve the trust problem of Section 3.1.11, the WebALPS guardian could itself verify the signatures of the posted content (using all the latest certificate revocation lists, etc.), and then include in the SSL-encrypted channel an assertion that this content had been verified, and the identity of its poster. The client Alice could then trust that content she downloaded via this SSL-encrypted channel from the trusted co-server did indeed originate with the alleged poster. (We can even save bandwidth here, since the client only need download the identity of the poster, not his public key, signature, and appropriate certificates.)

3.3.12 Solution: Integrity of Server Machine

To solve the trust problem of Section 3.1.12, the WebALPS guardian can witness that the appropriate computational security tool (such as a network security analyzer or a particular hardware-directed secure boot technique) was applied to the host—perhaps because this tool was applied from the co-server itself, or from a companion trusted machine. The client Alice can trust such assertions it receives through the SSL-authenticated communication channel from the co-server to the client.

4 Proving the Secure Coprocessing Thesis

Another compelling motivation for the WebALPS project is that provides a suitable arena for the next step in secure coprocessing research: demonstrating, in suitably concrete and real terms, that secure coprocessors solve difficult trust problems. Section 4.1 reviews this thesis; Section 4.2 presents the obstacles to demonstrating it; and Section 4.3 presents how WebALPS can overcome these obstacles.

4.1 The Thesis

In theory, one would think that the art had advanced to a state where secure coprocessors would immediately be addressing a wide variety of security problems. After all, the stage should be set:

- Our team's (and CMU's) earlier academic research demonstrated the potential of secure coprocessors.
- Our more recent industrial work transformed these devices from hand-built lab prototypes to mass-produced real-world products.

However, this technology still has not achieved its potential; to the computer community at large, we still need to establish the assertion:

> *Secure coprocessing solves security problems that would otherwise be difficult or impossible.*

4.2 Obstacles

After some consideration, we decided that the community has not been convinced because the paper designs and limited prototypes of the earlier academic work did not express the potential *in the proper language.*

Different audiences require different types of arguments to convince them. In order to convincingly demonstrate the potential of secure coprocessing, we need "arguments" that meet several key criteria:

- **Real Code** Paper designs are not sufficient. We need real functioning code, that is easy to access, easy to grasp, and has a professional user interface.

  Since this level of implementation for any one application will be a non-trivial amount of work, we need to identify a class of potential applications with common implementation tasks.
- **Important Problems** We need to use secure coprocessing to address problems that the audience cares about. In the current business and social climate, three areas in particular stand out:
  - *E-Commerce.*
  - *Privacy.*
  - *Moving computation into the network.*
- **Minimal Delta.** The secure coprocessing solutions we use to demonstrate our assertion need to be plausible to deploy on a wide scale in the real world. As a consequence, we need to minimize the delta between our deployed solution and the currently existing infrastructure.

  For example, any solution that starts out "imagine every citizen had a pocket-sized IBM 4758" will not be convincing.

4.3 Overcoming the Obstacles

In order to carry out the next step in our secure coprocessor research, we need a systematic way to demonstrate a wide family of applications that satisfies the criteria above. We believe that WebALPS can effectively demonstrate the potential of secure coprocessing, because it specifically overcomes the enumerated obstacles:

- **Real Code** By building real WebALPS demos accessible to anyone on the Internet[1], this work will be real and tangible.

  Preparation of the common framework–SSL key exchange and session manipulation—would provide a foundation to implement and demo a broad family of coprocessor applications.

- **Important Problems** Web-based services provide the perfect area to examine security and trust problems with immediate and graspable economic, social, and technical import.

  - **E-Commerce.** The existence of e-commerce fraud costs customers and merchants money. The potential for e-commerce fraud prevents merchants and customers from even entering the arena.
  - **Privacy.** As noted ad naseuem, moving business, government, and social services in the Internet explodes the potential for adversaries to learn, modify, and aggregate private information. WebALPS can address these concerns.
  - **Moving computation into the network.** By providing trusted neutral areas not at the nodes but further inside the network, WebALPS provides a way to secure computation that is moved away from these nodes—and also puts this security at a place that is easier to manage than the nodes.

- **Minimal Delta.** By using server-side SSL authentication, WebALPS can be deployed (at least for demo purposes) without changing the client infrastructure. (However, one cannot help but accumulate a wishlist for browser changes to enhance security and usability.)

  Furthermore, WebALPS essentially is free to client. Rather than adding expensive hardware to every client, WebALPS only requires adding hardware to the smaller set of service providers who already have a large equipment budget.

[1] We can give a new meaning to the acronym “DTSS”: the Dartmouth Trusted Secure Server.

5 Research Strategy

This paper draws a large picture: a project that can solve many manifestations of a real trust problem, and can constitute a convincing demonstration of the power of secure coprocessing.

However, for this project to achieve these goals, it has to move beyond a high-level paper design to real implementations, carefully considered. Section 5.1 considers some issues in mapping out this next line of work; Section 5.2 considers some tangential research issues that also emerged.

5.1 Main Thrust

Section 3.3 sketched a broad family of potential WebALPS applications (and brainstorming could easily produce more). One of the first tasks we need to do is to narrow down to one or two applications for initial implementation. Among other things, these applications need to be such that

- our implementation can easily demonstrate that the WebALPS solution is real, clear, and usable;
- measurements can show that the WebALPS solution does not overly impair integration with and performance of servers;
- and analysis/argument can show that these problems, when properly framed, could not easily have been solved another way.

One of the next steps is deciding *how* to implement WebALPS. At this point, it appears that we have two fundamentally different choices:

- run a complete Web server inside the 4758;
- bend a host Web server to:
  - push enough of SSL key exchange into the 4758, so that knowledge of the session key is confined within the coprocessor boundary;
  - route encrypted traffic into the 4758, where it will decrypt and do some primitive processing.

The former seems simpler in the long run—but is probably limited by the current software environment in 4758s (for now, developers only have access to a version of the CP/Q operating system; Linux has been ported to internal IBM prototypes, but is not available for external research yet).

Another step is deciding how the user interface for WebALPS should look. This area raises some surprising subtleties. For example:

- If the WebALPS session has some notion of an official windows from WebALPS, followed by a series of pages forwarded from the server, how does the user distinguish between the messages from WebALPS and messages from the server?
- If we do this with frames, how does WebALPS distinguish between responses to questions it asked, and responses to questions the potentially malicious server asked in its window?

A much less subtle issue is the fact that current browser technology, in all likelihood, does not meet the assumptions of Section 3.2.2.

- Continuing vulnerabilities with Javascript and other technologies make it doubtful that a portion of the screen could indeed be reserved for the exclusive rendering of authenticated WebALPS traffic.

- Most Web users probably are not even aware of what CAs their browser accepts, what this certification means, or what cryptography their browser has been configured to accept as suitable in SSL sessions.

  Expecting Web users to make easy judgments about whether they're talking with a WebALPS, and which type, will probably require a change to the browser.

- Experiments with the interaction of SSL sessions (and "locked locks" and warning windows) and frames suggest that, upon further investigation, no clear semantics will emerge about exactly when the lock is locked, across different browsers, different platforms, and different frame scenarios.

  Again, a browser change may be necessary.

5.2 Tangential Research

In addition to the main thrust of actually implementing the WebALPS framework and some applications, this work also suggests some additional research questions.

5.2.1 Trust Calculus

Why are some problems appropriate for secure coprocessors, and others not? It would be interesting to try to develop a more formal trust calculus (e.g., perhaps based on the sketches of Section 2.1.1) that could more precisely characterize which subset of security problems can be solved by this technology.

5.2.2 Cryptographic Protocols Between Parties with Asymmetric Resources

The security that SSL provides depends on the fact that only the client, and the server with the appropriate private key, can learn what the session key is. However, in the protocol as practiced, the session key is derived solely from data sent in the clear, or generated by the client. This means that an adversary who can observe network traffic and predict the client's PRNG can also generate the session key; potentially, such an adversary could pretend to be *any* certified server.

Upon further examination, this is not a problem with SSL per se, but a more generic problem of key exchange protocols (and possibly other protocols): the assumption that both parties involved have similar cryptographic and randomness resources. However, in the Web, this is not the case: a bona fide server will be much more likely to have good sources of randomness and high-speed crypto (e.g., because they can afford an IBM 4758) than a client. How to carry out protocols in such scenarios might be an interesting area of work (if no one's done it already).

For one starting point: in key exchange, compensating for weak client randomness does not require client authentication, if the client has secure storage.

- The client could generate or obtain (once) a keypair.
- The client then provides this uncertified public key in the initial part of the exchange.
- The server encrypts its parts of the exchange (that it usually sends in plaintext) with this public key.

The client still participates in providing entropy for the session key; but now the server's contribution is no longer in plaintext visible to the adversary. (However, one might argue that if the client has secure storage and enough good randomness to generate a keypair once, then they can build an unpredictable PRNG.)

5.2.3 Formal Models for Web Interaction

As part of this work, the author attempted—and finally postponed—to formally specify the security properties required for the data transmissions in WebALPS. The difficulty in formally specifying these properties stems in part from the difficulty in characterizing exactly what Web interaction is.

It would be interesting to build a formal model of this interaction. E.g., something like:

- Let $\mathcal{Q}$ and $\mathcal{H}$ be the sets of requests and responses.
- Each response H needs to map to some structured set of `html components`.
- For a particular Web service, some request Q_0 is designated as a valid initial request.
- A valid, completed Web session consists of an initial round of:
  - a request Q_0 from the user
  - a response $H_0 = F_{\text{server}}(Q_0)$

  followed by zero or more iterations of:
  - a request $Q_i = F_{\text{client}}(H_{i-1})$
  - a response $H_i = F_{\text{server}}(Q_i)$

F_{server} and F_{client} need to be defined appropriately, and include the influence of local state, etc.

This work would suggest some enlightening avenues of inquiry:

- modeling what the Web really does;
- defining various meanings of “secure” interaction;
- proving that none of these assertions can be true within this model; and
- defining various restrictions of the model, in which these “secure interaction” assertions can be true.

5.2.4 Meaningful Client Authentication

As we pointed out, another significant problem in the current Web infrastructure is the inability of service providers (except merchants, happy just to get a credit card number) to meaningfully authenticate the human at the client. The authentication subserver application partially addresses this problem. IBM’s announced “Trusted Client” work (very weak but very inexpensive secure coprocessors built into every new desktop) may provide a practical foundation to achieve this on a wider scale; this should be explored.

References

[1] J. Dyer, R. Perez, S.W. Smith, M. Lindemann. ‘Application Support Architecture for a High-Performance, Programmable Secure Coprocessor.” *22nd National Information Systems Security Conference.* October 1999.

[2] H. Gobioff, S.W. Smith, J.D. Tygar, B.S. Yee. “Smart Cards in Hostile Environments.” *2nd USENIX Workshop on Electronic Commerce.* 1996.

[3] W. Havener, R. Medlock, R. Mitchell, R. Walcott. *Derived Test Requirements for FIPS PUB 140-1.* National Institute of Standards and Technology. March 1995.

[4] *IBM PCI Cryptographic Coprocessor.* Product Brochure G325-1118. August 1997.

[5] *IBM Coprocessor First to Earn Highest Security Validation* Press release, IBM Corporation, December 1998.

[6] National Institute of Standards and Technology. *Security Requirements for Cryptographic Modules.* Federal Information Processing Standards Publication 140-1, 1994.

[7] E.R. Palmer. *An Introduction to Citadel—A Secure Crypto Coprocessor for Workstations.* Research Report RC 18373, IBM T.J. Watson Research Center, 1992.

[8] S. W. Smith. *Secure Coprocessing Applications and Research Issues.* Los Alamos Unclassified Release LA-UR-96-2805, Los Alamos National Laboratory. August 1996.

[9] S. W. Smith, V. Austel. “Trusting Trusted Hardware: Towards a Formal Model for Programmable Secure Coprocessors.” *The Third USENIX Workshop on Electronic Commerce.* September 1998.

[10] S.W. Smith, R. Perez, S.H. Weingart, V. Austel. “Validating a High-Performance, Programmable Secure Coprocessor.” *22nd National Information Systems Security Conference.* October 1999.

[11] S. W. Smith, E. R. Palmer, S. H. Weingart. “Using a High-Performance, Programmable Secure Coprocessor.” *Proceedings, Second International Conference on Financial Cryptography.* Springer-Verlag LNCS, 1998.

[12] S.W. Smith, D. Safford. “Practical Private Information Retrieval with Secure Coprocessors.” Research Report RC , IBM TJ Watson Research Center. July, 2000.

[13] S.W. Smith, S.H. Weingart. “Building a High-Performance, Programmable Secure Coprocessor.” *Computer Networks (Special Issue on Computer Network Security).* 31: 831-860. April 1999.

[14] S.H. Weingart. “Physical Security for the microABYSS System.” *IEEE Security and Privacy.* Oakland, 1987.

[15] S.R. White and L. Comerford. “ABYSS: A Trusted Architecture for Software Protection.” *IEEE Security and Privacy.* Oakland, 1987.

[16] S.H. Weingart, S.R. White, W.C. Arnold, G.P. Double. “An Evaluation System for the Physical Security of Computing Systems.” *6th Computer Security Applications Conference.* 1990.

[17] S.R. White, S.H. Weingart, W.C Arnold, E.R. Palmer. *Introduction to the Citadel Architecture: Security in Physically Exposed Environments.* Research Report RC 16672, IBM T.J. Watson Research Center, 1991.

[18] B.S. Yee. *Using Secure Coprocessors.* Ph.D. thesis. Computer Science Technical Report CMU-CS-94-149, Carnegie Mellon University. May 1994.

[19] B.S. Yee and J.D. Tygar. “Secure Coprocessors in Electronic Commerce Applications.” *1st USENIX Electronic Commerce Workshop.* 1996.