

Chapter 2

MOTIVATING SCENARIOS

In this chapter, we try to set the stage for our exploration of trusted computing platforms. In Section 2.1, we consider the adversary, what abilities and access he or she has, and what defensive properties a trusted computing platform might provide. In Section 2.2, we examine some basic usage scenarios in which these properties of a TCP can help secure distributed computations. Section 2.3 presents some example real-world applications that instantiate these scenarios. Section 2.4 describes some basic ways a TCP can be positioned within a distributed application, and whose interests it can protect; Section 2.5 provides some real-world examples. Finally, although this book is not about ideology, the ideological debate about the potential of industrial trusted computing efforts is part of the picture; Section 2.6 surveys these issues.

2.1 Properties

In its classic conception, a trusted computing platform such as a secure coprocessor is an armored box that does two things:

- It protects some designated data storage area against an adversary with certain types of direct physical access.
- It endows code executing on the platform with the ability to prove that it is running within an appropriate untampered environment.

What types of attacks the platform defends against, and exactly how code does this attestation, are issues for the platform architect.

In an informal mental model of a distributed computing application, we map computation and data to platforms distributed throughout physical space. Users (including potential adversaries) are also distributed throughout this space. Co-location of a user and a platform gives that user certain types of access to

that platform: through “ordinary” usage methods as well as malicious attack methods (although the distinction between the two can sometimes reduce to how well the designer anticipated things). A user can also reach a platform over a network connection. However, in our mental model, direct co-location differs qualitatively. To illicitly read a stored secret over the network, a user must find some overlooked design or implementation flaw in the API. In contrast, when the user is in front of the machine, he or she could just remove the hard disk.

Not every user can reach every location. The physical organization of space can prevent certain types of access. For example, an enterprise might keep critical servers behind a locked door. Sysadmins would be the only users with “ordinary” access to this location, although cleaning staff might also have “ordinary” access unanticipated by the designers. Other users who wanted access to this location would have to take some type of action—such as picking locks or bribing the sysadmins—to circumvent the physical barriers.

The potential co-location of a user and a platform thus increases the potential actions a user can take with that platform, and thus increases the potential malicious actions a malicious user can take. The use of a trusted platform reduces the potential of these actions. It is tempting to compare a trusted platform to a virtual locked room: we move part of the computation away from the user and into a virtual safe place. However, we must be careful to make some distinctions. Some trusted computing platforms might be more secure than a machine in a locked room, since many locks are easily picked. (As Bennet Yee has observed, learning lockpicking was standard practice in the CMU Computer Science Ph.D. program.) On the other hand, some trusted computing platforms may be less secure than high-security areas at national labs. A more fundamental problem with the locked room metaphor is that, in the physical world, locked rooms exist before the computation starts, and are maintained by parties that exist before computation starts. For example, a bank will set up an e-commerce server in a locked room before users connect to it, and it is the bank that sets it up and takes care of it. The trusted computing platform’s “locked room” can be more subtle (as we shall discuss).

2.2 Basic Usage

This discussion leaves us with the working definition: a TCP moves part of the computation space co-located with the user into a virtual locked room, not necessarily under any party’s control. In more concrete terms, this tool has many potential uses, depending on what we put in this separate environment. At an initial glance, we can look on these as a simple 2x2 taxonomy: secrecy and/or authenticity, for data and/or code.

Since we initially introduced this locked room as a data storage area, the first thing we might think of doing is putting data there. This gives *secrecy of data*. If there is data we do not want the adversary to see, we can shelter it in the

TCP. Of course, for this protection to be meaningful, we also need to look at how the data got there, and who uses it: the implicit assumption here is that the code the TCP runs when it interacts with this secure storage is also trustworthy; adversarial attempts to alter it will also result in destruction of the data.

In Chapter 1, we discussed the difference between the terms “trustworthy” and “trustable”. Just because the code in the TCP might be trustworthy, why should a relying party trust it? Given the above implicit assumption—tampering code destroys the protected data—we can address this problem by letting the code prove itself via use of a key sheltered in the protected area, thus giving us *authenticity of code*.

In perhaps the most straightforward approach, the TCP would itself generate an RSA key pair, save the private key in the protected memory, and release the public key to a party who could sign a believable certificate attesting to the fact that the sole entity who knows the corresponding private key is that TCP, in an untampered state. This approach is straightforward, in that it reduces the assumptions that the relying party needs to accept. If the TCP fails to be trustworthy or the cryptosystem breaks, then hope is lost. Otherwise, the relying party needs only needs to accept that the CA made a correct assertion.

Another public key approach involves having an external party generate the key pair and inject the private key, and perhaps escrow it as well. Symmetric key approaches can also work, although the logic can be more complex. For example, if the TCP uses a symmetric key as the basis for an HMAC to prove itself, the relying party must also know the symmetric key, which then requires reasoning about the set of parties who know the key, since this set is no longer a singleton.

Once we have set up the basis for untampered computation within the TCP to authenticate itself to an outside party—because, under our model, attack would have destroyed the keys—we can use this ability to let the computation attest to other things, such as data stored within the TCP. This gives us *authenticity of data*. We can transform a TCP’s ability to hide data from the adversary into an ability to retain and transmit data whose values may be public—but whose authenticity is critical.

Above, we discussed secrecy of data. However, in some sense, code is data. If the hardware architecture permits, the TCP can execute code stored in the protected storage area, thus giving us *secrecy of code*. Carrying this out in practice can be fairly tricky; often, designers end up storing encrypted code in a non-protected area, and using keys in the protected area to decrypt and check integrity. (Chapter 6 will discuss this further.) An even simpler approach in this vein is to consider the main program public, but (in the spirit of Kerckhoff’s law) isolate a few key parameters and shelter them in the protected storage.

However, looking at the potential taxonomy simply in terms of a 2x2 matrix overlooks the fact that a TCP does not just have to be passive receptacle

that holds code and data, protected against certain types of adversarial attack. Rather, the TCP houses computation, and as a consequence of this protected environment and storage, we can consider the TCP as a computational entity, with state and potentially aware of real time. This entity adds a new column to our matrix: rather than just secrecy and authenticity, we can also consider *guarding*. Whether a local user can interact with the stored data depends on whether the computational guard lets him or her; whether a local user can invoke other computational methods depends on whether the guard says it is permissible.

2.3 Examples of Basic Usage

Secrecy of Data. An axiom of most cryptographic protocols is that only the appropriate parties know any given private or secret key. Consequently, a natural use of TCPs is to protect cryptographic keys. A local user Bob would rather not have his key accessible by a rogue officemate; an e-commerce merchant Alice would rather not have her SSL private key accessible by an external hacker or a rogue insider.

Authenticity of Code. Let's continue the SSL server example. Bob might point his browser to Alice's SSL server because he wants to use some service that Alice advertises. The fact that the server at the other end of the Internet tunnel proved knowledge of a private key does not mean that this server will actually provide that service. For example, Bob may wish to whisper his private health information so Alice's server can calculate what insurance premium to charge him; he would rather Alice just know the premium, rather than the health information. For another example, perhaps Alice instead is a healthcare provider offering an online collection of health information. Bob might wish to ask Alice for a record pertaining to some sensitive disease, and he would rather no one—not even Alice—know which topic he requested.

In both these cases, Bob wants to know more than just that the server on the end of the tunnel knows the private key—he also wants to know that the server application that wielded this data and provides this service actually abides by these privacy rules.

Authenticity of Data. Suppose instead that Alice participates in a distributed computation in which she needs to store a critical value on her own machine. For example, we can think of an “e-wallet” where the value is the amount cash the wallet holds, or a game in which the value is the number of points that Alice has earned. We might even think more generally: perhaps this value is the audit log of activity (potentially from hackers) on Alice's machine.

In all these situations, the value itself might reasonably be released to Alice and to remote parties (under the appropriate circumstances). However, in these situations, parties exist who might have access to this value, and might have

motivation to alter it. Alice may very well have motivation to increase her wallet and point score; an attacker who's compromised Alice's machine might very well want to suppress or alter the audit log. The remote party wants assurance that the reported value is accurate and current.

Secrecy of Code. Despite textbook admonitions against “security through obscurity,” scenarios still arise in the real world where the internal details of a program are still considered proprietary. For example, credit card companies use various advanced data mining approaches to try to identify fraudulent account activity and predict which accounts will default, and regard the algorithm details as closely held secrets. Similarly, insurance companies may regard as proprietary the details of how they calculate premiums based on the information the applicant provided.

If Alice is such a party, then she would not want to farm her code out to Bob's site unless Bob could somehow assure her that the details of the code would not leak out. In this case, the TCP enables an application that otherwise might not be reasonable.

Guarded Data. In the e-wallet case above, Alice's TCP holds a register indicating how much money Alice's wallet holds. Consider how this value should change: it should only increase when the e-wallet of some Bob is transferring that amount to it; it should only decrease when Alice's e-wallet is transferring that amount to the e-wallet of some Bob. In both these situations, the exchange needs to be fully *transactional*: succeeding completely or failing completely, despite potential network and machine failures.

In this case, the relying party needs to do more than just trust that the value allegedly reported by Alice's e-wallet was in fact reported by Alice's e-wallet. Rather, the relying party also needs to be able to trust that this value (and the values in all the other e-wallets) has only changed in accordance with these transactional rules. By providing an authenticated shelter for code interacting with protected data, a TCP can address this problem.

For another case, consider an electronic object, such as a book or a movie, whose usage is governed by specific licensing rules. For example, the book may be viewed arbitrarily, but only on that one machine; the movie might have the additional restrictions of being viewed only N complete times, and only at ordinary speed. In both cases, a TCP could store the protected data (or the unique keys necessary to decrypt it), as well as house a program that uses its knowledge of state and time to govern the release of the protected object.

Of course, for this technology to be effective against moderately dedicated attackers, either the TCP needs to have an untappable I/O channel to release the material, or the material that is released during ordinary use must be somehow

inappropriate for making a good pirated copy. (For one examples, we could use the TCP to insert watermarks and fingerprints into the displayed content.)

The notion of a protected database of sensitive information—where stakeholder policy dictates that accesses be authorized, specific, and rare—satisfies this latter condition. One example of such a database might be archives of network traffic, saved for later use in forensic investigation.

Guarded Code. As a natural extension to the above DRM example, we could change the book to a program—since the assumption that the adversary would not reverse-engineer the program solely from the I/O behavior observed during normal use is far more reasonable. In this case, the guard would prevent the program from operating—or migrating out of the TCP—unless these actions comply with the license restrictions. For the case in which the TCP is too limited in computational power to accommodate the program it is intended to protect, researchers have *proposed partitioned computation*: isolating a critical piece of the program that is hard to reverse-engineer, and protecting that piece inside the TCP.

A more trivial example would be a cryptographic accelerator: we do not want the TCP to just store the keys; we also want it to use the keys only when properly authorized, and only for the intended purpose. (As recent research shows, doing this effectively in practice, for current cryptographic hardware supporting current commodity PCs, is rather tricky.)

2.4 Position and Interests

Putting trusted computing protections in place for something that occurs only in one place involving one party does not achieve much. Arguably, TCPs only make sense in the context of a larger system, distributed in space and involving several parties. In the current Internet model, the initial way we think of such a system is as a local client interacting with a remote server. Typically, these terms connote several asymmetries: the client is a single user but the server is a large organization; the client is a small consumer but the server is a large content provider; the client handles rather little traffic, but the server handles much; the client has a small budget for equipment, but the server has a large one.

TCPs need to exist in a physical location, and to provide a virtual island there representing the interests of a party at another location. Initially, then, we can position a TCP in two settings:

- at the client, protecting the interests of the server,
- or at the server, protecting the interests of the clients.

However, like most things initial, this initial view misses some subtleties.

- Sometimes, a TCP at Alice's site can advance her own interests, much as a bank vault helps a bank. The TCP can help her protect her own computation against adversaries and insider attack. In e-commerce scenarios, this protection can even give her a competitive advantage.
- The client-server model may indeed describe much distributed computation. However, it does not describe all of it: for example, some systems consist instead of a community of peers.
- Naively, we think of a TCP as protecting some party's interests. However, the number of such parties does not necessarily have to be one.
- Naively, we also think of a TCP providing a protected space that extends the computational space controlled by some remote party. However, the number of parties who "control" the TCP's protected space does not necessarily have to be nonzero. E.g., if Alice is to reasonably gain a competitive advantage by putting some of her computation into a locked box, then the locked box must be subsequently under *no one's* control.

2.5 Examples of Positioning

Client-side. The standard DRM examples sketched above constitute the classic scenario where the TCP lives at the client side and protects the interests of a remote server (in this case, the content provider). The operator of the local machine would benefit from subverting the protections, in order to be able to copy the material or watch the movie after the rental period has expired. Symmetrically, the remote content provider would (presumably) suffer from this action, due to lost revenue.

Server-side. Above, we also sketched examples where the TCP lived at the server side:

- enforcing that access to archived sensitive data follows the policy agreed to before the archiving started; or
- providing a Web site where clients can request sensitive information, without the server learning what was requested.

These cases invert the classic DRM scenario. The TCP now lives at the server side and protects the client's interests by restricting what the server can do.

Protecting own interests. This privacy-enhanced directory application also inverts the standard model, in that the TCP at the server side also arguably advances the server's interests as well: the increased assurance of privacy may draw more clients (and perhaps insulate the server operator against evidence

discovery requests). Another example would be an e-commerce site that provides gaming services to its clients, and uses a TCP to give the clients assurance that the gaming operations are conducted fairly. By using the TCP to provide a space for fair play, the server operator advances her own interests: because more clients may patronize a site that has higher assurance of fairness.

We can also find examples of this scenario at the client. Consider the problem of an enterprise whose users have certified key pairs, but insist on using them from various public access machines, exposed to potential compromise. In one family of solutions, user private keys live in some protected place (such as at a remote server, perhaps encrypted). When Alice wishes to use her private key from a public machine, she initiates a protocol that either downloads the key, or (in one subfamily) has the machine generate a new key pair, which the remote server certifies.

In these settings, Alice is at risk: an adversary who has compromised this public machine can now access the private key that now lives there. However, suppose this machine used one of the newer TCP approaches that attempt to secure an entire desktop. We could then amend the key protocol to have the remote server verify the integrity of the client machine before transferring Alice's credential—which helps Alice. Thus, by using a TCP at the client to restrict the client's abilities, we advance the interests of the client.

Multiple parties. As we observed, the parties and protected interests involved can be more complex than just client and server. Let's return the health-insurance example. Both the client and the insurance provider wish to see that an accurate premium is calculated; the client further wishes to see that the private health information he provided remains private. Using a TCP at the insurance provider thus advances the interests of multiple parties: both the client and the server. We can take this one step further by adding an insurance broker who represents several providers. In this case, any particular provider might farm out her premium-calculation algorithm to the broker, but only if the broker can provide assurances that the details of the algorithm remain secret. So, a TCP at the broker now advances the privacy interests of both the consumer and the external provider, the accuracy interests of all three parties, and the competitive advantage of the broker.

For another example, consider the challenges involved in carrying out an online auction. Efficiency might argue for having each participant send in an encoding of his or her bidding strategy, and then having a trusted auctioneer play the strategies against each other and announce the winner. However, this approach raises some security issues. Will the auctioneer fairly play the strategies against each other? Will the auctioneer reveal private details of individual strategies? Will the auctioneer abide by any special rules advertised for the auc-

tion? Can any given third party verify that the announced results of an auction are legitimate?

We could address these issues by giving the auctioneer a TCP, to house the auction software, securely catch strategies, and sign receipts attesting to the input, output, and auction computation. The TCP here protects the interests of each of the participants against insider attack at the auction site and (depending on how the input strategies are packaged) against fraudulent participant claims about their strategies.

Community of peers. Consider the e-wallet example from earlier. If Bob can manage to increase the value of cash his e-wallet stores without going through the proper protocol, then he essentially can mint money—which decreases the value of everyone’s money. In this case, the TCP at a client is protecting the interests of an entire community of peer clients.

Of course, the classic instantiation of such community-oriented systems is *peer-to-peer* computation: where individual clients also provide services to other clients, and (often) no centralized servers exist. Investigating the embedding of TCPs in P2P computation is an area of ongoing research. For example, in distributed storage applications that seek to hide the location and nature of stored items, using TCPs at the peers can provide an extra level of protection against adversaries. For another example, the *SEmi-trusted Mediator (SEM)* approach to PKI breaks user private keys into two pieces (using *mediated RSA*), and stores one piece at a trusted server, who (allegedly) only uses it under the right circumstances. We could gain scalability and fault tolerance by replacing the server with a P2P network; using TCPs at the peers would give us some assurance that the key-half holders are following the appropriate rules.

No one in control. As we discussed above, in a naive conception, the TCP provides an island that extends the controlled computational space of some remote party. However, note that a large number of the above applications depend on the fact that, once the computational entity in the TCP is set up, no one has control over it, not even the parties whose interests are protected. For example, in the private information server, neither the server operator nor the remote client should be able to undermine the algorithm; in the auction case, no party should be able to change or spy on the auction computation; in the insurance broker case, the insurance provider can provide a premium calculation algorithm that spits out a number, but should not be able to replace that with one that prints out the applicant’s answers.

How to build a TCP that allows for this sort of uncontrolled operation—while also allowing for code update and maintenance—provides many challenging questions for TCP architecture.

2.6 The Ideological Debate

The technology of trusted computing tends to focus on secrecy (“the adversary cannot see inside this box”) and control (“the adversary cannot change what this box is doing”). Many commercial application scenarios suggested for this technology tend to identify the end user as the adversary, and hint at perhaps stopping certain practices—such as freely exchanging downloaded music, or running a completely open-source platform—that many in our community hold dear.

Perhaps because of these reasons, the topic of trusted computing has engendered an ideological debate. On the one side, respected researchers such as Ros Anderson [Anda] and activist groups such as the Electronic Frontier Foundation [Sch03b, Sch03a] articulate their view of why this technology is dangerous; researchers on the other side of the issue dispute these claims [Saf02b, Saf02a for example].

Any treatment of TCPs cannot be complete without acknowledging this debate. In this book, we try to focus more on the history and evolution of the technology itself, while also occasionally trying to show by example that TCP applications can actually be used to empower individuals against large wielders of power.

2.7 Further Reading

We’ll consider many of these applications further in Chapter 4, Chapter 9, and Chapter 11.



<http://www.springer.com/978-0-387-23916-3>

Trusted Computing Platforms

Design and Applications

Smith, S.

2005, XX, 239 p., Hardcover

ISBN: 978-0-387-23916-3