Automating the Software Development Process

Jamie Floyd
University of Virginia
jfloyd@cs.virginia.edu

Abstract

This paper surveys the development and use of the spi calculus, an extension of the pi calculus with cryptographic primitives. It motivates the creation of the initial pi calculus and demonstrates how it can be used to describe abstract security protocols. However, the basic pi calculus has limitations when attempting to model realistic protocols, so the spi calculus was developed to help address them. Both the pi and spi calculus are extremely expressive abstract programming languages that also can offer proofs about certain security properties. This paper shows an example of a simply security protocol in both languages. It then discusses how the spi calculus is used in research and practice today.

Categories and Subject Descriptors D.2.0 [*Security*]: general

Keywords concurrency, security

1. Introduction

A significant area of work within programming languages research is the development of high-level languages that abstractly model computing systems in a useful way. The most well known example is likely the λ-calculus, which is a functional programming language that is equally powerful as a Turing machine. While it excels at modeling sequential programs, the λ-calculus and its variants are insufficient at modeling concurrent systems. The pi calculus, however, is capable of modeling these systems very effectively. Among many others, one major application of the pi calculus is describing security protocols. Unfortunately, the base pi calculus has some limitations when trying to describe a realistic security protocol. Abadi and Gordon sought to extend the pi calculus with cryptographic primitives to better enable the modeling of these systems. They called their result the spi calculus.

This paper surveys the major works in the development and use of the spi calculus. Section 2 introduces the pi calculus, without which understanding the spi calculus would be impossible. It details the motivation, history, and basic primitives of the language. Section 3 then introduces the spi calculus, noting the major differences from the pi calculus. A simple example is run through both sections 2 and 3. Section 4 presents usages of the spi calculus in recent years, and section 5 concludes.

PLDI '14, Dec 12, 2014.

http://dx.doi.org/10.1145/

2. The Pi Calculus

2.1 History

When computer networks were first created and came into widespread use, it became clear than communicating systems are fundamentally different than previous computer systems. Similarly, the tools we had to model computing systems, such as the well-known λ calculus, failed to capture many of the important properties of communication in a distributed world. Consequently, researchers begin looking for other ways of describing communicating systems. The resulting tools were known as process calculi, or methods of calculating or modeling concurrent systems through processes.

The first major process calculus was introduced by Hoare in 1978 [1]. It was known as CSP (Communicating Sequential Processes). Just two years later, Robin Milner proposed a similar approach called CCS (the calculus of communicating systems) [2]. The CCS had a stronger formal basis than CSP and was more flexible

as well. It was a well-accepted tool, but had one major flaw: it was limited to static communication structures. It described communication over a specific, static network of communication channels. It was impossible in CCS to add, remove, or alter channels in the system.

The pi calculus is an extension to CCS proposed by Milner in 1989 that is designed to address that flaw [3]. It replaced static communication channels with channel names. Instead of using an established static system of channels, the pi calculus introduced the notion of a dynamic system of named channels and communication instructions that depended on channel names. Channels can be created dynamically, assigned to variables, and passed as messages.

The resulting system was still elegantly simple but extremely expressive. All functional programs can be encoded into the pi calculus and this encoding emphasizes the dialog nature of concurrent computation. Just as the λ calculus is a foundation for sequential computation, the pi calculus is a foundation for concurrent computation.

2.2 Outline of the Pi Calculus

First, it is worth noting that there are multiple versions of the pi calculus. This paper will outline the same version as the original spi calculus paper, simplified for brevity.

Much like everything in the λ calculus is a function, everything in the pi calculus is a process. The set of processes in the pi calculus is defined by the grammar:

```
P,Q,R ::=
    M[N].P              output
    M(x).P              input
    P | Q               composition
    (vn)P               restriction
    !P                  replication
    [M is N] P          match
    0                   nil
```

The constructs of the pi calculus have the following intuitive meanings:

- An output process $m[N].P$ is ready to output on channel m. If an interaction on that channel occurs, the term N is communicated and then process P runs.
- An input process $m(x).P$ is ready to input from channel m. If an interaction occurs in which N is communicated on m, then process $P[N/x]$ runs (P with every free occurrence of x replaced with N).
- A composition $P|Q$ behaves as processes P and Q running in parallel. Each may interact with the other or with other channels independently of each other.
- A restriction $(\mu n)P$ is a process that makes a new private name n which may occur in P then behaves as P. It is important to note that the scope of n is *restricted* to P.
- A replication $!P$ behaves as an infinite number of copies of P running in parallel.
- A match $[MisN]P$ behaves as P provided that terms M and N are the same. Otherwise, it does nothing.
- The nil process does nothing.

There is one more term the needs an intuitive definition for use int the rest of this paper: $P \simeq Q$ means the behaviors of processes P and Q are indistinguishable. They may have different internal structure, but a third process R cannot distinguish running in parallel with P from running in parallel with Q. While [4] defines this relation in further detail, this informal understanding will suffice for this paper's scope. Anytime this paper references the equivalence or indistinguishability or two processes, this notion applies.

2.3 Example Using the Pi Calculus

As mentioned before, it is possible to describe some abstract security protocols purely in the pi calculus. Common security protocols are designed around channels on which only a given set of parties are allowed to send or receive data. As we have seen, this is incredibly simple in the pi calculus.

In this example, two parties, A and B, share a channel c_{AB}. Only A and B can send data or listen through this channel. The protocol is simply that A uses c_{AB} to send a single message M to B. In the pi calculus:

$$A(M) \triangleq c_{AB}[M]$$

$$B \triangleq c_{AB}(x).F(x)$$

$$Inst(M) \triangleq \mu c_{AB}(A(M)|B)$$

The processes $A(M)$ and B describe the two parties and $Inst(M)$ describes an instance of the whole protocol. The effect of this instance is: restricted channel c_{AB} is created, A sends M over c_{AB}, B receives M

from c_{AB}, binds it to local variable x, then computes F on x.

The protocol described above has two important cryptographic properties:

1. Authenticity or integrity: B always applies F to the message M that A sends. An attacker cannot cause B to apply F to some other message.
2. Secrecy: the message M cannot be read in transit from A to B: if F does not reveal M, then the whole process does not reveal M.

Both of these properties can be defined in terms of process equivalence. The secrecy property is simple: if $F(M) \triangleq F(M')$ for any M, M' then $Inst(M) \triangleq Inst(M')$. Another way to say this is if $F(M)$ is indistinguishable from $F(M')$, then the protocol with message M is indistinguishable from the protocol with message M'. To formalize the authenticity property, compare this protocol $Inst(M)$ with another protocol, defined as follows:

$$A(M) \triangleq c_{AB}[M]$$

$$B_{spec}(M) \triangleq c_{AB}(x).F(M)$$

$$Inst_{spec}(M) \triangleq \mu c_{AB}(A(M)|B_{spec}(M))$$

In this protocol, A is as usual. However, B is replaced with a variant $B_{spec}(M)$ that receives the same input and acts like B but $B_{spec}(M)$ already knows M. We take $Inst_{spec}(M) \triangleq Inst(M)$ for any M as our authenticity property.

3. The Spi Calculus

3.1 Motivation

As we have seen, it is possible to describe a security protocol, however primitive, in the pi calculus. However, the pi calculus is not capable of describing every protocol, particularly more complex ones.

From the modern cryptographer's perspective, the primary lacking of the base pi calculus is the specification that channels are restricted when they are created with no further explanation. It is natural to wonder how we could ever claim that a channel is completely private, and that no adversary would be able to get any information from it (or, in fact, know that it exists!). One possible expansion to handle this is to describe in further detail how to create such a channel that would be completely hidden. However, this is not how most of modern cryptography works. Instead, we can say that all communication channels are public and just pass data through it that would be of no use to an adversary. Namely, we want to encrypt and decrypt data and only have the ciphertext available on the public channel. This is the approach of the spi calculus, which is built as an extension of the pi calculus with cryptographic protocols.

3.2 Outline of the spi calculus

The spi calculus requires two primary additions to the syntax of the pi calculus. First, to represent encryption, $\{M\}_N$; second, to represent decryption, $case\ L\ of \{x\}_N\ in\ P$.

The intuitive meanings of these terms is as follows:

- The term $\{M\}_N$ represents the ciphertext obtained by encrypting the term M under the key N using a shared-key cryptosystem.
- The process $case\ L\ of \{x\}_N\ in\ P$ attempts to decrypt the term L with the key N. If L is a ciphertext of the form $\{M\}_N$, then the process behaves like $P[M/x]$. Otherwise it does nothing.

There are some significant cryptographic assumptions implicit in this definition:

1. The only way to decrypt a ciphertext is to know the corresponding key (note that, as of the original spi calculus, there is no notion of probability so this statement is an absolute).
2. An encrypted ciphertext does not reveal the key that was used to encrypt it.
3. There is sufficient redundancy in messages so the decryption algorithm can detect if a ciphertext was encrypted using the expected key.

3.3 Example Using the Spi Calculus

This section will continue the example from section 2.3, extending and modifying it with cryptographic primitives. The essence of the protocol will be very simple: we assume A and B share a private key k_{AB} and there exists a public channel c_{AB} on which A and B can communicate. A will send message M to B under k_{AB} on c_{AB}. In the spi calculus, we write

$$A(M) \triangleq c_{AB}[\{M\}_{k_{AB}}]$$

$$B \triangleq c_{AB}(x).case\ x\ of \{y\}_{k_{AB}}\ in\ F(y)$$

$$Inst(M) \triangleq \mu k_{AB}(A(M)|B)$$

where $Inst(M)$ is once again an instance of the protocol. This is very similar to the example in the pi calculus, but with a couple key differences.

- We now have the notion of encryption, decryption, and ciphertexts. Note that these are not specific encryption or decryption algorithms - this is a model of a generic private shared key cryptosystem. As such, it is only as secure as the cryptosystem itself.
- Our notion of security no longer relies on the strict restriction of a channel. The channel c_{AB} in this example is completely public. The only requirement is that both A and B be able to use it; otherwise it has no security guarantees.

Our notions of authenticity and secrecy are very similar. Secrecy remains completely unchanged: $Inst(M) \triangleq Inst(M')$ if $F(M) \triangleq F(M')$ for any M, M'. To formalize the notion of authenticity, we once again create another protocol to compare to $Inst(M)$:

$$A(M) \triangleq c_{AB}[\{M\}_{k_{AB}}]$$

$$B_{spec}(M) \triangleq c_{AB}(x).case\ x\ of \{y\}_{k_{AB}}\ in\ F(M)$$

$$Inst_{spec}(M) \triangleq \mu k_{AB}(A(M)|B_{spec}(M))$$

Then authenticity is defined as $Inst(M) \triangleq Inst_{spec}(M)$ for any M.

3.4 Other Examples

In [3], Abadi et. all give several other examples of protocols in both the pi and spi calculi. For instance, the model a simplified version of the "wide mouthed frog protocol" that allowed for the creation of a new restricted channel from A to B through some trusted party S. The spi calculus turned this protocol into one of key agreement through a trusted third party. [3] also goes on to describe an authentication protocol in the spi calculus, but it is omitted here for brevity.

3.5 Discussion

Even given these limited protocols, it is clear that writing formally in the spi calculus is more difficult than other informal notation systems. However, the spi calculus versions are more detailed; they make clear what messages are sent, how those messages are generated, and how they are checked. These benefits of the spi calculus come with a complexity price, but enable finer analysis of the protocols they describe.

4. The Spi Calculus in the 21st Century

The spi calculus was originally developed in 1996 (though not published until the next year). It was received well at the time, but only as a formal modeling language for security protocols. With more than 15 years to develop, it is natural to wonder what has come from the spi calculus that is benefiting the field today.

There has been recent work [5][6] that uses recent advances in automatic program generation or synthesis to construct real implementations of security protocols based on a spi calculus specification. The authors of [6] note that the spi calculus is an untyped high level programming language. They propose the addition of a type system as well as a translation function from spi calculus to Java. They, as an extension of [5], add formal conditions on a custom Java library such that if the library implementation code satisfies the conditions then the generated Java program correctly simulates the spi calculus specification. This paper proposes only part of the necessary library to make this verified translation a reality.

The motivation to continue that work is strong, however. The authors point out the many advancements in automatically verifying security properties in the spi calculus, pointing to work done by Durante, Blanchet, and Abadi since the turn of the century. Through a combination of model checking and theorem proving, we are able to verify many security properties on protocols described in the spi calculus. [5] describes work on an automatic tool to detect protocol flaws based on only spi calculus specifications.

5. Conclusions

The pi calculus is a process calculus created to model concurrent systems that can dynamically alter their communication channels. It is possible to model some security protocols in the pi calculus, but its expressiveness and utility is limited. To enhance it's usefulness for security, the pi calculus was extended with cryptographic primitives in the spi calculus. The spi calculus is effective at describing security protocols at a high level. It is possible to automatically check certain security properties of protocols written in the spi calculus, and some work has been done to automatically convert spi calculus to Java implementations while retaining those security guarantees.

References

[1] C.A.R. Hoare. Communicating Sequential Processes, in ACM 1978.

[2] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, I and II, in ScienceDirect, 1989.

[3] M. Abadi, A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus, in Conference on Computer and Communications Security (CCS), 1997.

[4] M. Abadi, A. Gordon. Reasoning about Cryptographic Protocols in the Spi Calculus, in CONCUR, 1997.

[5] D. Pozza, R. Sisto, L. Durante. Spi2Java: Automatic Cryptographic Protocol Java Code Generation from spi calculus, in International Conference on Advanced Information Networking and Application (AINA), 2004.

[6] A. Pironti, R. Sisto. Provably correct Java implementations of Spi Calculus security protocols specifications, in ScienceDirect, 2008.