Adding capability-based security to high performance Parallex

Habraken, J.A.E.

Award date:
2013

Master's thesis

Adding capability-based security to High Performance ParalleX

Jeroen Habraken

Supervisor:	dr. Jerry den Hartog
Tutor:	dr. Hartmut Kaiser
Committee:	dr. Jerry den Hartog dr. Hartmut Kaiser dr. Benne de Weger dr. Nicola Zannone

Abstract

The first exa-scale computers are predicted to arrive in 2018 but we are currently unable to fully utilise such massive parallelism. ParalleX is an execution model which attempts to solve this through hiding system-wide latencies, decoupling hardware execution resources from executing software tasks, and enabling runtime dynamic adaptive scheduling of these tasks. In order for this model to be evaluated High Performance ParalleX (HPX) has been developed, the first feature-complete, open-source implementation of ParalleX.

In this thesis we design a capability-based protection system for HPX through the collection of requirements from scenarios and subsequently using these to select the primitive building blocks to secure the system. This allows a machine with read capabilities to monitor a simulation without influencing it for example. To prove the design its viable a partial implementation of the design is done within HPX. Afterwards the cost of the protection layer is evaluated by comparing unprotected and protected runs as increased latencies are expected due to the cryptographic overhead.

Acknowledgements

First of all I would like to thank Hartmut Kaiser from the Center for Computation and Technology for the opportunity of conducting this research abroad. Your pragmatical approach has been inspiring. My gratitude goes to the whole team, including the STE||AR group, at the Louisiana State University who made the visit possible, and the numerous new friends I've made. These include Bryce Adelstein-Lelbach, Vinay Amatya, Zach Byerly, Adrian Serio, and last but not least my roommate Masoud Safari-Zanjani.

I would also like to thank Jerry den Hartog, my supervisor here at the Eindhoven University of Technology , without whom this text could not have been written. A thanks also goes out to all my fellow students with whom I have enjoyed all these years.

Additionally I would like to thank the people who have proofread my thesis and provided feedback, especially Agustín Bergé.

Finally I would like to thank my parents and sisters for their love and support, and their patience when frustrations ran high. This leaves me to quote Winston Churchill, as it so beautifully captures the moment.

> Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.
>
> — *Winston Churchill*

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

High Performance Computing (HPC) is the use of powerful computers, high-bandwidth low-latency networks, large capacity fast storage, parallel filesystems and specialised parallel software to provide answers to specific scientific, engineering, and societal questions.

It is crucial to a wide variety of research fields, it supports the modelling and investigation of natural processes at all scales, from the atomistic and molecular through to astronomical. Weather forecasts, to name an example, are the output of such a HPC model and critical to aviation amongst other things. [29]

Security within these heterogeneous, distributed settings is essential for two reasons. First of all due to the fact that the compute cluster comprises a vast amount of powerful resources, which when abused could wreak havoc to the internet. This could range from a simple Distributed Denial of Service (DDoS) attack to distributed password cracking, or a complete botnet running on a compute cluster.

More importantly, many of the aforementioned research fields deal with sensitive data, either with regard to intellectual property or privacy. Examples of this are in the medical field where HPC is used for drug discovery, or in finance where it is used to calculate risk values for use in High Frequency Trading (HFT).

Security and performance have always been at odds with each other though, which leads to a conflict of interest. This is due to the fact that performance is, as the name implies, essential to HPC after all whilst security on the other hand comes at a cost of performance.

Given the new ParalleX execution model [14] we were curious whether access control within the High Performance ParalleX runtime, and thus applications, was viable.

1.2 Objectives

The foundation of any master's thesis is the research question, and this one is constructed based on the following.

> Is it viable to do access control within high performance computing, leveraging the ParalleX execution model to meet the security requirements whilst minimising the performance impact?

We have formulated several objectives to be able to answer this research question. First we aim to find typical usage scenarios from which we can extract security, performance, and scalability requirements. These scenarios will then also be used as a guideline when designing an access control system that should meet these requirements.

Finally we aim to validate this high level design by creating a proof of concept implementation and evaluating its performance and scalability. Ultimately we wish to determine the feasibility of real world usage, and estimate which problems can be expected scaling the solution to the ever growing computing solutions.

1.3 Structure of the thesis

First of all we introduce the ParalleX execution model in chapter 2, which serves as the basis of the High Performance ParalleX runtime which we describe in chapter 3. Subsequently chapter 4 introduces the required security background.

Chapter 5 then covers the scenarios, narratively at first, and then analyses these to construct a formal set of requirements. These are used as a guide in the high level design which is presented in chapter 6, after which chapter 7 highlights several parts of the proof of concept implementation.

In chapter 8 the high level design is evaluated, using both the security and performance requirements which were previously formalised. We conclude this thesis with our conclusions and suggestions for future work in chapter 9.

CHAPTER 2

The ParalleX Execution Model

In HPC parallel efficiency and speedup refer to how much faster a parallel application is than its corresponding sequential implementation, or in other words, how well the application scales. Two common notions of scaling exist, strong and weak scaling. The former, strong scaling, is defined as the time variance of a fixed size problem when varying the amount of parallel resources available. Weak scaling on the other hand is defined as the time variance of a variable sized problem on an amount of parallel resources that are varied proportionally to the problem.

For an application to utilise as much parallelism as possible it has to support both strong and weak scaling, requiring a large part of the application to be executed in parallel. In the optimal case an application scales linearly, thus running N times faster or handling N times more data when executed on N processors instead of 1. In practice this is merely a theoretical limit though as scalability is limited by current hardware architectures and executing models. The limiting factors can be summarised by the acronym *SLOW* [31]:

*S*tarvation: Insufficient concurrent work to maintain high utilisation of resources.

*L*atencies: Time-distance delay of remote resource access and services.

*O*verhead: Work for management of parallel actions and resources on the critical path which are not necessary in the sequential variant.

*W*aiting for contention: Delays due to lack of availability of oversubscribed shared resources.

ParalleX tries to improve efficiency by reducing the average synchronisation and scheduling overhead, improve utilisation through asynchrony of workflow, and employ adaptive scheduling and routing to mitigate contention. Scalability will be increased, at least for certain classes of problems, through data directed computing using message-driven computation and lightweight synchronisation mechanisms that will exploit the parallelism intrinsic to dynamic directed graphs through their meta-data. These techniques will be described in more detail in the following sections.

As a consequence, sustained performance will be improved both in absolute terms through extended scalability for those applications currently constrained, and in relative terms due to enhanced efficiency achieved. Finally, power reductions will be achieved by reducing extraneous calculations and data movements. [13]

2.1 Latency hiding

Whilst it is impossible to design a system exposing zero latencies, efforts have been made to approach this as closely as possible through optimisations. Low-latency network technologies such as InfiniBand, and caching memory hierarchies in processors [27] are just a few examples of such optimisations.

These latencies are often introduced by resources having to wait for an operation to be completed, slowing down the application. Instead this time could be spent on unrelated tasks, allowing the latencies to be hidden by filling the idle-time with useful work. A similar technique is already being employed in modern processors in the form of pipelined instruction execution. [17]

Latency hiding is an intrinsic concept of the ParalleX execution model, utilising asynchronous operations throughout the whole system stack.

2.2 Fine-grained parallelism

The ideal mode of operation to support the above latency hiding would be to schedule asynchronous tasks in a queue. These tasks might be very short-lived, such as fetching the uncached contents of a memory cell from main memory for example. To support such short tasks we need very lightweight threads with extremely short context switching times, optimally executable within a single processor cycle. Whilst this is not possible with today's architectures it is described in the ParalleX execution model and a long-term goal. The basic concept still holds for the architectures we have today though, the smaller the

overhead of a context switch and the finer the granularity of the threading system, the better the overall system utilisation and its efficiency.

Several libraries such as Intel's Thread Building Block (TBB) and Microsoft's Parallel Patterns Library (PPL) are already providing these non-preemptive, task-queue based parallelisation solutions. They suspend a task if some precondition for its execution is not met, like waiting for input, output, or the result of a different task. At that point it is possible to seamlessly switch to another task which can continue, and reschedule the initial task to continue after the required precondition has been met.

2.3 Constraint-based synchronisation

The synchronisation of control flow between several, or often all, threads or processes of an application is generally handled by a global barrier. For instance, a global barrier is inserted after each loop parallelised using OpenMP, an execution model for shared memory systems. This barrier implicitly synchronises the threads used to execute the different iterations in parallel as shown in figure 2.1. Even minimal fluctuations in the execution time of the tasks run in these threads causes them to unnecessarily wait on each other.

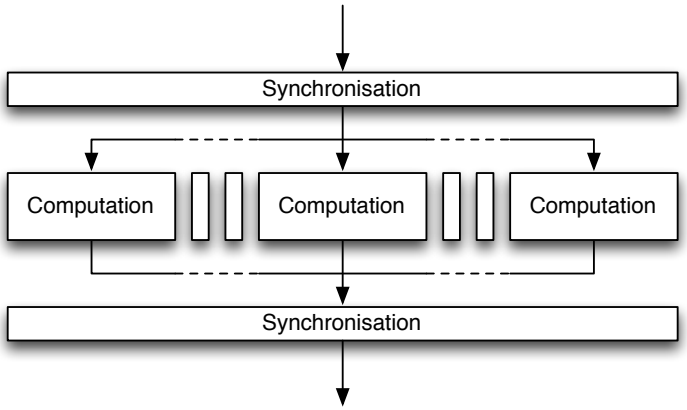


Figure 2.1: Typical control flow of a loop with a global barrier

An analysis of several key algorithms used in science show these global barriers are not always necessary [6], and in many cases it is sufficient to synchronise a small subset of the threads. Tasks should proceed whenever their preconditions are met, and only those, which is precisely what the ParalleX model proposes. There is no need to wait for all iterations

of a loop to finish before continuing for instance, you only need to have those iterations done which were producing the required results for a particular next task.

2.4 Adaptive locality control

The current execution models leave it to the developer of an application to decompose the data to be processed and distributed over the localities the application is running on. This makes it very tedious to distribute the data in an adaptive and dynamic way.

To overcome this limitation the Partitioned Global Address Space (PGAS) was developed, together with a couple of specialised languages and programming environments such as Chapel [4], X10 [5], UPC [3], and Co-Array Fortran [26]. However, all PGAS-based systems rely on static data distribution, which only works if it does not result in resource utilisation imbalances such as the workload being distributed inhomogeneously. In a distributed system these imbalances can be mitigated by migrating a part of the application data to different localities but at the moment the only framework supporting limited migration is Charm++.

The ParalleX model defines the Active Global Address Space (AGAS) which incorporates the notions of a global, possibly distributed, uniform address space and adds the capability of data migration to flexibly support adaptive and dynamic locality control. AGAS is further discussed in section 3.3.

2.5 Moving work to the data

When running an application on a distributed memory machine it is inevitable that data in the form of bytes has to be transferred from one part of the system to another. For best performance the amount of bytes transferred should be minimised on all levels. At the lowest level this means taking advantage of processor memory caches, minimising memory latencies, whilst at the higher level we should minimise the data transfer between localities. This minimises network latencies as moving larger amounts of data back and forth introduces larger overheads.

In general the amount of bytes necessary to encode an operation is smaller than the bytes encoding the data the operation is to be performed on. Nevertheless we typically still transfer the data to where the operation is to be executed only to transfer the result back to the source afterwards. The ParalleX execution model intends to put an end to this using parcels [14] which represent an action to be executed. The implementation of these parcels are described in section 3.4.

2.6 Message driven computation

Today's prevalently used programming model on distributed systems is Message Passing Interface (MPI), which, as the name implies, is based on message passing. This means the receiver has to be aware of a message about to come in, implicitly requiring the sender and receiver to synchronise in order to perform a communication step. As a result a more than trivial MPI application generally spends a considerable amount of time waiting for incoming messages, causing starvation and latencies to impede full resource utilisation. The more complex and more dynamic the algorithms and data structures become, the larger the adverse effects.

Message driven computation on the other hand allows messages to be sent without the receiver actively having to wait for them. Incoming messages are handled asynchronously and trigger the encoded action by passing along the arguments. ParalleX combines this scheme with the task-queue based scheduling described in section 2.2. This allows it to almost completely overlap any communication with useful work, reducing latencies to a minimum.

2.7 Summary

The techniques described above all aid ParalleX in achieving its key aims, the first of which is to to expose new forms of program parallelism to increase the total amount of concurrent operations.

Secondly it aims to reduce overheads improving efficiency of operation and, in particular, to make effective use of fine-grain parallelism where it should occur. This includes, where possible, the elimination of global barriers.

Last of all it aims to facilitate the use of dynamic methods of resource management and task scheduling to exploit runtime information about the execution state of the application and permit continuing adaptive control for best causal operation. [13]

CHAPTER 3

The High Performance ParalleX runtime

This section introduces the High Performance ParalleX (HPX) runtime [14], a feature-complete research implementation of the above ParalleX execution model in C++, designed for both optimal performance and portability. Figure 3.1 shows its architecture and highlights several of the essential components which will be described in the sections below.

3.1 Localities

Every instance of the HPX runtime is a locality, effectively mapping the notion of a locality to the system-specific concept of a conventional process. Multiple instances may be started on a single compute node and these will be traditionally managed by the underlying operating system. However, as HPX is able to fully utilise the compute power of a node a single instance will generally be started on each node within a commodity cluster. This is to prevent instances from having to share resources with each other. Through the AGAS and parcel transport services described in sections 3.3 and 3.4 respectively HPX will automatically coordinate these instances to present a single system view to the application.

HPX allows localities to join and part this single system view at runtime, which is useful if more compute nodes become available whilst the application is running, or are required

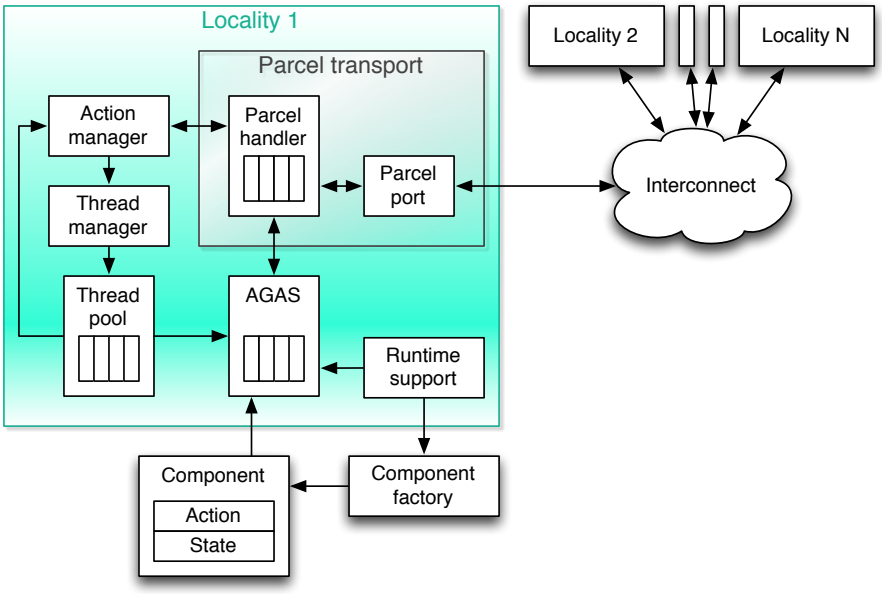


Figure 3.1: The High Performance ParalleX runtime architecture

elsewhere.

3.2 Thread management

Lightweight threads and their efficient scheduling are crucial to ParalleX, and thus to the HPX runtime, as described in section 2.2. The lightweight user-space thread implementation in HPX makes it possible for the scheduler to manage millions of threads. Note that contrary to Operating System (OS) threads these are cooperative, thus they need to voluntarily suspend execution, which is inherent to being implemented in user-space.

The user-space scheduler maps these lightweight threads on to the available OS threads, the number of which is generally determined by the underlying hardware. The default implementation of this scheduler uses a single lock-free FIFO queue per OS thread. When such a queue is empty the scheduler has the ability to steal tasks from other OS-threads' queues, efficiently load balancing the work. [24] The efficiency of this scheduler is also achieved by avoiding calls into the underlying OS kernel, which are generally expensive. This means the HPX application can run for the whole time slice provided by the OS

scheduler.

3.3 Active Global Address Space

Every instance of a named object in HPX is assigned a 128-bit Global Identifier (GID) by the AGAS service, which manages the 128-bit address space spanning all localities. It currently runs on the first locality and is available to other localities via the parcel transport service described in section 3.4. Local caching of GIDs minimises the number of network round-trips, and thus overhead.

The service consists of two components, the first of which maintains a mapping between a GID, and a locality and Local Virtual Address (LVA). As shown in Figure 3.2 the GID is split up into the Most Significant Bits (MSB) and Least Significant Bits (LSB) which together comprise the 32-bit locality identifier, the reference count, and the component identifier which is a unique 80-bit number per component within the locality. The reference count acts as a garbage collection scheme preventing an object from being destroyed until it is out of scope on all localities.

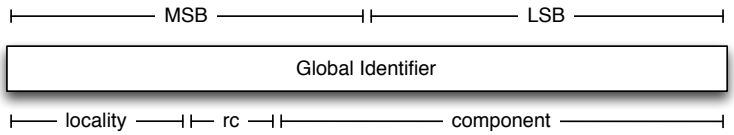


Figure 3.2: The structure of a Global Identifier

The second, higher level component maintains a mapping between symbolic names and GIDs, and is used by the performance counters.

3.4 Parcel transport

As HPX is message driven each locality runs the parcel transport service which handles inter-locality messaging based on parcels. This service too consists of two components; the parcel port and the parcel handler.

The parcel port abstracts the network interface and is responsible for sending and receiving parcels via several concrete implementations to handle shared memory, TCP/IP, and InfiniBand for instance. When a parcel is received it is passed to the parcel handler which translates the parcel into an action to be executed by the action manager. These actions are described below in section 3.5.

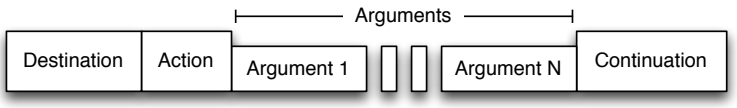


Figure 3.3: The structure of a parcel

The parcels themselves consist of four fields as shown in Figure 3.3. The GID of the destination, the action to be executed, the arguments to pass to the action on execution, and the GID of a continuation. This continuation is a callback action to be executed on completion, and to which the result of the action is passed.

3.5 Actions and components

An action is a special type in HPX which wraps a C++ function, defining its name, and type information such as the argument and return types. Wrapping a function in an action allows it to be transported using the parcel transport described above and executed as a ParalleX thread on other localities, essentially providing a Remote Procedure Call (RPC) service. Two types of actions exist, const and non-const where the former does not mutate the system state whereas the latter does.

A second distinction is made between the action types, there being plain actions which wrap global functions and component actions which wrap component member functions. As HPX is a C++ library it borrows core constructs from the language which we can see with actions as well as components, modelling functions and classes respectively. These classes are globally named, meaning they can be referenced from any locality through their GID, which allows the localities to share state. The components expose their methods through component actions allowing them to be called remotely as well.

Within HPX each locality has a special component named the runtime support which provides system services such as component creation. The runtime support system calls into a component factory which in turn creates the component. This indirection allows the components to be created remotely by other localities.

CHAPTER 4

Security

The Confidentiality, Integrity, and Availability (CIA) triad as shown in figure 4.1 is one of the fundamental concepts of information security. Each of the three concepts is generally easy to implement by itself, but their contradicting nature makes it challenging to find a balance when combining all three.

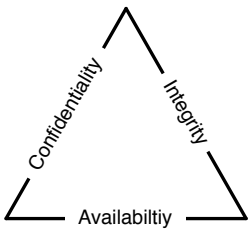


Figure 4.1: The Confidentiality, Integrity, and Availability triad

Confidentiality is the preservation of authorised restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. The main mechanisms used in the protection of confidentiality are access controls and cryptography.

Integrity is defined as guarding information against improper modification or destruction. These safeguards may be grouped into two broad types, preventive mechanisms, such as access controls that prevent unauthorised modification of information, and detective mechanisms, which are intended to detect unauthorised modifications when preventive mechanisms are not an option or have failed.

Availability, last of all, means ensuring timely and reliable access to and use of information. [9] It is trivial after all to secure information in such a way that confidentiality and integrity are guaranteed, one can simply lock it in a vault and dispose of the key. This, however, is useless as the information is not available to authorised users when, and where, it is needed and hence the aforementioned balance must be found between all concepts in the triad.

In this chapter we will use the CIA framework to situate the security related concepts and technologies that we will be introducing, starting with access control.

4.1 Access control

As stated above access control is an important mechanism used to protect both confidentiality as well as integrity. Accessing information is often done through a three stage process, identification, authentication, and authorisation, each of which will be discussed in the following sections.

Access control, subsequently, is the process of deciding which subject has what access rights on which objects with respect to some security models and policies.

4.1.1 Identification

Without the possibility to uniquely identify subjects many additional security mechanisms will fail. This is due to the fact that subjects could spoof the identity of other subjects, or create an arbitrary number of additional identities.

An identity is a reference to a subject, which may be a person or something else entirely. The identity of a subject is a unique property that is tied irreversibly to this single subject, it should be unchangeable throughout the lifetime of the subject, and not be transferred to other subjects. [15]

Identities generally have a large number of properties associated with them, some of which may be used as identifiers. In the context of a person as shown in figure 4.2 this may include details such as their legal name, what their face looks like, the sound of their voice, their reputation for various functions, and possibly a public key. When given an unknown

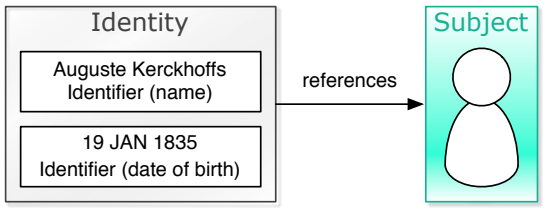


Figure 4.2: A subject referenced by an identity with identifiers

person's legal name you can, with a moderate degree of confidence, check whether any of the identities you have stored can be identified using the name.

This is not always possible though, a given identifier may be ambiguous, or of insufficient fidelity to reliably identify a unique person. A first name for example may not be unique enough to identify a single person but it may rule out others. To summarise, an identifier is a property (or group of properties) which is suited to identify a subject, i.e. doubtlessly determine its identity according to the properties defined above. [15]

4.1.2 Authentication

Authentication, then, is the act of verifying to a high degree of confidence that the party you are communicating with really is the party you believe it is, and not some impostor or usurper. This is usually done through the validation of a secret key ("something you know"), the use of hard-to-forge identifiers ("something you are"), which is the foundation of biometrics for instance, validation of a token ("something you have") such as a drivers license, or a combination of the above. [23]

Validation of a secret key, classically a password, passphrase, or Personal Identification Numbers (PIN), is the most commonly used method as it it generally cheap and easy to implement.

4.1.3 Authorisation

Authorisation, last of all, consists of two separate processes. The first of which is assigning a set of authorisations that define what a subject can, and can not do on a system after declaring their identity at the identification stage and proving it at the authentication stage. At the same time, authorisation is the process of ensuring that a subject has sufficient rights to perform a requested operation and preventing those without sufficient rights from doing the same.

The first of these processes is guided by a security policy that outlines how information is accessed, what level of security is required, and what actions should be taken when these requirements are not met. A policy outlines the expectations of a computer system or device whilst a security model is a statement that outlines the requirements necessary to properly support and implement a certain security policy. In the popular Bell-LaPadula model [20] for example, each subject is assigned a clearance level and each object is attached a security level like top secret or classified.

Many systems use Lampson's access matrix [19] to represent and interpret a particular security policy. In such a matrix the rows represent subjects and columns represent objects as shown in table 4.1. The access rights that a subject holds for an object can be found at the intersection of the row and the column belonging to the subject and the object respectively. As the number of objects in the system grows it becomes complex to manipulate the matrix directly, and in practice they tend to be very sparse thus so most systems do not store the access rights in a matrix form. They rather use either an access control list approach or a capability approach.

	$object_1$	$object_2$	$object_3$
$subject_1$	read	read, execute	
$subject_2$		read, write, execute	read, write

Table 4.1: An example of an access control matrix

In the access control list approach, the matrix is viewed by column. Each object is associated with an access control list which stores the subjects and their access rights for the object. The list is checked to see whether to grant an access. In the capability approach on the other hand, the matrix is viewed by row. Each subject is associated with a capability list which stores its access rights to all concerned objects, and possessing a capability is the proof of possessing the corresponding access rights. [11]

4.2 Cryptographic systems

Two distinct types of cryptographic systems exist, the first of which is a secret-key cryptographic system, also known as symmetric key system, in which the same key is used for both the encryption of a plaintext and the decryption of a ciphertext. Thus the contents of an encrypted message can be obtained by anyone who is in possession of the secret key used to encrypt the message. Advanced Encryption Standard (AES) and Data Encryption Standard (DES) are examples of such secret-key ciphers.

A public-key system on the other hand, also known as asymmetric key system, is so named because different keys are used for the encryption and decryption operations. The encryption key is referred to as the public key whilst the decryption key is referred to as

the private key. Even if an attacker possesses the public key used to encrypt a message it is unable to obtain the plaintext without the corresponding decryption key, which should hence be kept private. Here it is important to note that these systems, such as RSA for example which is described in section 4.2.1, are built in such a way that the private key can not be deduced from the public key.

The private key in the above system can be considered a token, with a subject able to prove to be in possession of the private key matching a public key without revealing it, allowing it to be used to authenticate a subject. The public-key variant of the Needham–Schroeder–Lowe protocol for example is built on top of this, and allows mutual authentication of two subjects. [25, 21]

Secret-key cryptographic operations are generally computationally less expensive, and the key lengths shorter, than public-key cryptographic operations of equivalent robustness. Therefore most practical public key exchanges are used to establish an ephemeral shared secret key between communicating parties, which is then used to protect subsequent communications. [23]

4.2.1 RSA

RSA is an algorithm for public-key cryptography, which, as public-key cryptography in general, is based on the intractability of a certain mathematical problem. In the case of RSA it is the presumed difficulty of factoring large integers, the factoring problem. The algorithm is named after Ron Rivest, Adi Shamir and Leonard Adleman who first publicly described the algorithm in 1977. [28]

To use RSA the product of two large random prime numbers is computed and published along with an auxiliary value as the public key. These two prime factors must be kept secret as they form the basis of the private key. Anyone can use this public key to encrypt a message, but with currently published methods only someone with knowledge of the prime factors, assuming the public key is large enough, can feasibly decode the message. As of writing a 2048-bit public key is considered large enough, but 3072-bit or larger keys are advisable.

4.2.2 Elliptic curve cryptography

Elliptic curve cryptography (ECC) is also based on the intractability of a certain mathematical problem. Here it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible, with the size of the elliptic curve determining the difficulty of the problem.

The primary benefits of ECC are that is it less computationally expensive and has a smaller key size reducing storage and transmission requirements. An elliptic curve group can provide the same level of security afforded by an RSA-based system with a large modulus and correspondingly larger key. A 256-bit ECC public key for example provides comparable security to a 3072-bit RSA public key.

4.3 Man in the Middle attacks

If an authentic channel, a channel resistant to tampering but not necessarily resistant to overhearing, is available the two parties could simply send copies of their public keys to each other. These public keys could then be used as the basis for a cryptographic scheme to guarantee integrity such as those described in section 4.5.1 and 4.5.2.

However, such an authentic channel is not always available, and exchanging the public keys over an untrusted channel is inherently unsafe. An attacker with the ability to intercept messages and inject them into the untrusted channel can execute a simple attack whereby they impersonate the intended recipient during the public key exchange. Without the ability to determine whether the public key received over the untrusted network is authentic the initiating party A might not be sending their confidential messages M, such as an ephemeral shared secret, to the intended recipient but directly to an attacker E as shown in figure 4.3.

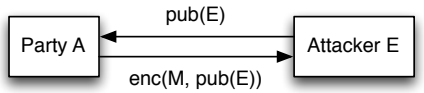


Figure 4.3: Messages exchanged in an impersonation attack

When the attacker extends the simple attack described above with a second attack against party B, impersonating A, this this can be made more convincing as it is then able to relay messages between the two parties. In relaying them it is able to read and manipulate the plaintext before forwarding it without either party being able to detect it. This extended attack is called a Man in the Middle (MITM) attack and is shown in figure 4.4. The implication of it is that it is generally considered impossible to establish a secure channel over an untrusted network without a shared secret or a trusted public key available to allow the parties to mutually authenticate each other.

Using a shared secret or certified public key, which will be discussed in detail in the next section, it is possible for two parties to establish such a mutually authenticated channel. Even an attacker with the ability to modify messages crossing the network will lack the

Figure 4.4: Messages exchanged in an Man in the Middle attack

secret key, and is thus unable to synthesise messages that will be accepted as authentic by the parties on either side. [23]

4.4 Certificate schemes

In all certificate schemes every identifiable subject has their own key pair to identify itself to other subjects in a public-key cryptographic system as discussed in section 4.2. The best approach depends on the number of subjects in the system.

In a small system each subject can exchange their public key with the other subjects with which they wish to interact. This is effective for communicating with a few subjects which you known and have met before, but breaks down when you want to communicate securely with a subject you have not met before, or when the number of subjects becomes intractably large.

Larger, more sophisticated systems employ distributed certification schemes that allows a Trusted Third Party (TTP) to assert the veracity of others subjects' public keys. This allows two parties who have not previously exchanged public keys over an authenticated channel to still authenticate and communicate securely with each other if their respective public keys have been certified by the TTP.

Such a TTP issues signed certificates, which bind subjects' names to their respective public keys. This allows the credentials to be verified offline, without the involvement of the TTP for every public key exchange. However, these extra layers of indirection come at a cost, the reliance on additional trusted systems increases the number of places where security can be compromised. [23]

The sections below will describe three different models for certification systems.

4.4.1 Certificate authority

In the Certificate Authority (CA) model only the authorities are trusted to make identity assertions about subjects. Their function is to reliably evaluate claims of ownership of an identity that is provided to them, generally though a Certificate Signing Request (CSR),

and to issue cryptographically-signed certificates to authentic claimants. These certificates typically bind an identity to a public key that was presented in the original CSR.

Therefore, as hinted at in section 4.1.2 the certificate, coupled with proof from an unauthenticated party that they hold the private key that corresponds to the public key in their certificate, can be regarded as a strong claim by the unauthenticated party that they are truly who they claim to be.

However, this claim can be undermined by either the CA not being trusted to make such assertions or the loss or theft of any of the private keys involved. To mitigate the latter type of failure a CA will generally maintain a revocation list of keys which have been cancelled.

These components together, sometimes referred to as a Public-Key Infrastructure (PKI), are typically used to support authentication across untrusted networks. In such a system, every authenticating agent must maintain a set of trust anchors, certificates that identify the set of CAs which they trust to assert the identity of other subjects correctly.

Generally the CAs are configured in a hierarchy, as shown in figure 4.5. Rather than signing certificates directly they instead issue certificates to one or more intermediate authorities. Unlike certificates issued to subjects, intermediate authorities' certificates are specially marked by their parent as being authorised to assert the identity of others on its behalf. [23]

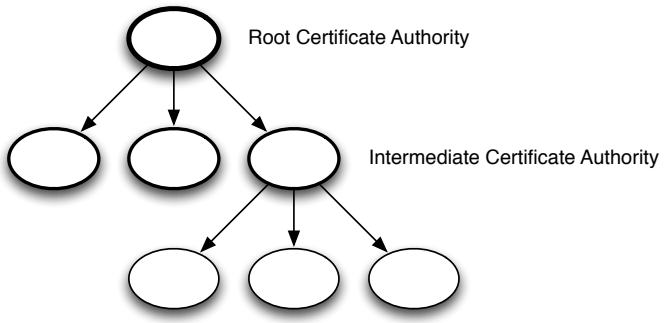


Figure 4.5: A Certificate Authority hierarchy

4.4.2 Identity-based cryptography

Identity-Based Cryptography (IBC) is a technique which was first proposed by Adi Shamir in 1984. [30] A binary representation of the identity is used to derive the public key of a subject, which can then be used in an public-key cryptography scheme. This inherently

solves the problem in the above case where the CA was required to link the identity and public key.

The subject then authenticates to a Key Generation Centre (KGC), a TTP which generates a matching private key from a secret that it owns. Note that this private key must be acquired over a secure channel, a channel that is resistant to overhearing and tampering. This is a stronger requirement than in the PKI case above because it concerns a private key which must not be overheard.

Subjects can subsequently authenticate any other subject as their public key can be deduced from their identity.

4.4.3 Web of trust

Another technique is the so called web of trust. Subjects can mutually sign their respective keys expressing their belief that the key belongs to the subject its identity claims to belong to. Instead of a TTP verifying the identities the subjects can do this themselves, building, as the name implies, a web of trust. Subjects can assign trust to other subjects' keys expressing belief in their ability to correctly verify identities and apply signatures to other keys. Whether a subject believes a signature depends on the trust level calculated from this trust, which may be transitive.

This is significantly different from the above models in that every subject, not just those blessed with a CA or KGC status, may certify the credentials of another. However, those subjects are not automatically accepted as trusted to certify the identity of another. Every subject must independently decide exactly which other subjects it will trust to make identity assertions, and in turn how much those assertions will be trusted.

For example, if subject A uses its key k_A to sign the key k_B of subject B it expresses its believe that k_B belongs to B, and similarly when B signs the key of C. Now if a subject D trusts A to create valid signatures it can validate the key k_B being the key of subject B by checking the signature from A as shown in figure 4.6. If it trusts A transitively it can even validate k_C after validating k_B.

4.5 Hash functions

A hash function H is a function which maps an arbitrary-length input i to a fixed-length series of bytes h, often called the hash value. Several different types of hash functions exist with different properties, but here we will describe cryptographic hash functions which have three important properties.

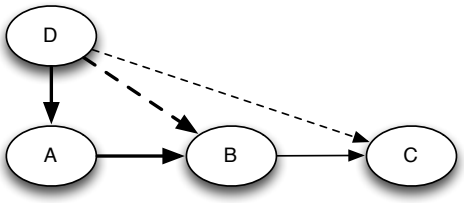


Figure 4.6: A web of trust

First of all a cryptographic hash function must be preimage resistant, which means that given a hash value h it must be computationally infeasible to find an input i such that $H(i) = h$.

Secondly it must be collision resistant meaning that it must be infeasible to find two different inputs i_1 and i_2 such that $H(i_1) = H(i_2)$. A second part to the collision resistance is chosen-prefix collision resistance, thus given two different prefixes p_1 and p_2 it must be infeasible to find two appendages i_1 and i_2 such that $H(p_1|i_1) = H(p_2|i_2)$ with $|$ the concatenation operator.

Last of all they must be second-preimage resistant meaning it must be computationally infeasible to find any second input which has the same hash value as any specified input. In other words given i_1 it must be infeasible to find a second preimage i_2 with $i_1 \neq i_2$ such that $H(i_1) = H(i_2)$.

A fourth property is often named, the ability to cheaply compute the hash value of any given input.

These cryptographic hash functions have several uses, two of which are intended to detect unauthorised modifications of an input message and are described below.

4.5.1 Message authentication codes

A Message Authentication Code (MAC) is a short code used to provide integrity and authenticity assurances on the message. The integrity assurances detect accidental and intentional changes to the message whilst the authenticity assurances affirm the messages' origin. The MAC algorithms accept an arbitrary-length input message and a secret key to output the MAC. These algorithms are generally based on block ciphers or hash functions, and due to the latter are often called keyed hash functions. [18]

Figure 4.7 shows the MAC protocol where depending on the comparison of the two MACs the received message is accepted or rejected. The authenticity as well as the integrity

assurances are derived via the secret key. Any subject which is in possession of the shared key is able to generate a valid MAC thus one of them must be the sender, assuming the key has not been compromised. An attacker on the other hand is not able to generate a valid MAC after modifying the message as it is not in possession of this key. The key itself may be derived from the authorisation process, such as the ephemeral shared secret mentioned in section 4.1.2, or explicitly established afterwards using the Diffie-Hellman key exchange for example. [7]

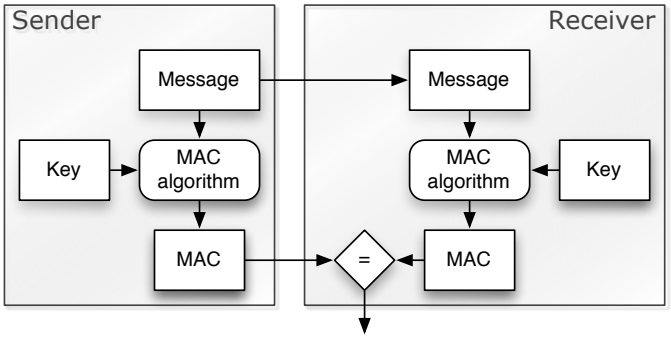


Figure 4.7: The MAC protocol

4.5.2 Signatures

Whilst a digital signature is in many ways similar to its analog equivalent, showing an input message to originate from a certain sender, it also assures the integrity of the message.

To sign an input message it is cryptographically hashed, after which the hash value is encrypted using the private key of the sender. The hash value alone provides an integrity assurance but an attacker would be able to modify the parcel and compute a new hash value. After encryption everyone is able to decrypt the signature using the public key of the sender, however, as an attacker does not have access to the private key of the sender it is unable to compute a new signature.

Verifying a signature first requires the receiver to obtain the certificate of the sender and verify its validity using one of the techniques we have shown in section 4.4. If this certificate is valid the receiver can extract the public key, decrypt the signature, hash the input message, and compare this hash value to that in the decrypted signature. Successively, when they are equal, it may accept the message.

CHAPTER 5

Scenarios and goals

This chapter will narrate two scenarios to describe the potential use of access control within HPX. These will be analysed to extract requirements which will subsequently be formalised to guide us when constructing the high level design in chapter 6. Finally the proof of concept implementation will be tested against these requirements in an evaluation presented in chapter 8

5.1 Scenarios

5.1.1 Monitoring

The simplest distributed setting in which HPX is often used is a commodity computing cluster, a cluster of heterogeneous machines interconnected via a shared network. As described in section 3.1 a locality would be started on a subset of the nodes within the cluster to execute a set of computations, such as the simulation of a model as mentioned in the introduction.

The ability to connect a locality outside of the cluster to monitor the status, and potentially other variables within the simulation, as shown in figure 5.1 would be useful to several people. First and foremost the user who started the simulation, to monitor its progress,

aid in debugging, and prematurely abort it when it is heading in the wrong direction for example.

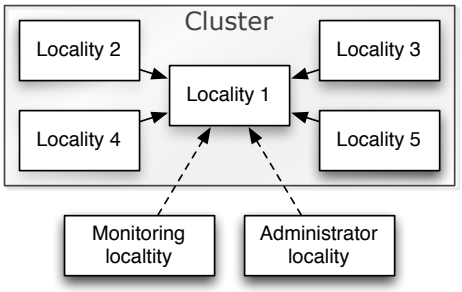


Figure 5.1: An illustration of the monitoring scenario

It would also be useful to the cluster administrators who are interested in the simulation progress as this would allow them to better schedule the work on the cluster, increasing its utilisation. However, the cluster users do not want their administrator to be able to manipulate their simulation, even accidentally. Thus the monitoring locality needs the ability to read the current status of the work but should not be able to mutate the actual work being done.

5.1.2 Distributed graph

Facebook, LinkedIn, and a multitude of other companies are built around social graphs which have become increasingly important in our everyday lives over the last few years. These types of graphs are generally very large in size, Facebook for instance currently has 1.11 billion users, thus it is efficient to only keep a single instance of this graph in memory and allow different applications, or components within an application to manipulate it.

Similar to a more conventional database not every operation needs the same rights. The algorithm to find a node its neighbours, or find the shortest path between two nodes does not need to mutate the graph for example, whereas others will. This can be modelled by an HPX application which distributes the graph information through a component exposing certain actions. Successively allowing different functional components to interact and operate on this information component within their capabilities.

In the simplest case the distinction can be made between components which do or do not mutate the graph, but this can be extended to partition the information. This could be used to only allow certain components access to a subset of the graph which contains sensitive data for example.

5.2 Analysis

The objective of this is to add an access control system to HPX applications, only allowing authorised localities to join and participate in the computation. In the monitoring scenario this security needed to be purely at the locality level, whereas in the distributed graph scenario the rights differed between components. Ultimately it should only allow subjects to execute the component actions that is has permission for. The access control itself will need to be designed generically such that authors of an application can shape it to their needs.

Secondly, whilst the network within a cluster, and indirectly its administrators, are generally trusted the monitoring machine in the respective scenario was connected from outside of the cluster. This means the content of that connection, whilst not confidential, must be protected against manipulation. An attacker should not be able to manipulate this content to execute different actions with the rights of the monitor, or an action with different arguments than intended. Also, connecting a monitoring locality should not interfere with the application running on the cluster its availability, that should continue undisturbed.

Last of all we are in the HPC domain thus performance and scalability are obviously very important. This is also shown by ParalleX, and thus HPX, its key aim to reduce overheads to improve efficiency of operation. Whilst additional overhead is inevitable when introducing new features and functionality we should minimise the overhead where possible. Valid content should be delivered to its destination and executed with as little delay as possible, without introducing problems that would undermine the availability of information to authorised subjects. Nor should we impede scalability, the exact problem that HPX is trying to solve.

5.3 Goals

The following subsections will formally enumerate the security and performance requirements extracted from the scenarios through the analysis above.

5.3.1 Security requirements

Security is our primary concern in this design, which is captured by the functional requirements blow which need to be incorporated into the design in order to have access control in HPX.

SR-1 A locality shall be authenticated before remotely executing actions.

As we have seen in chapter 4 security evolves around trust, and within HPX we want to bootstrap this trust from the user and then inductively expand on it. In remotely executing actions a locality essentially takes this inductive step, and as such starts participating in an application at which point we need it to be authenticated.

SR-2 Access control shall be added to prevent a component from executing actions it is not allowed to.

SR-3 The access control mechanism shall be flexible enough to adapt to the application developers' needs.

In the scenarios and analysis above we have seen the reasoning behind the flexibility, this formally captures it in a requirement.

SR-4 A connecting locality shall not impede the application its availability.

The CIA triad in chapter 4 showed the balance required between confidentiality, integrity, and accessibility. Through this requirement we prevent the scales from being tipped and only confidentiality and integrity being taken into account.

SR-5 The parcels' integrity shall be guarded when sent over an untrusted network.

Parcels contain the action to be executed and the arguments to pass to the action on execution as shown in section 3.4. When these are sent over an untrusted network an attacker could potentially modify them to execute a different action or execute an action with different arguments than intended.

5.3.2 Performance requirements

Whilst security is our primary concern performance is a close second given its essential to HPX as explained in chapter 1.

PR-1 No more than 250 microseconds of overhead shall be added in dispatching a parcel.

As shown when introducing the ParalleX execution model latencies are one of the four limiting factors as captured by the *SLOW* acronym. In discussions with an expert 250 microseconds was determined to be the maximum amount of overhead that would be acceptable.

PR-2 No more than 5 percent overhead shall be added to an application by the access control.

As with the above requirement the 5 percent level was determined in discussion with an expert. At this percentage it is barely perceived by users.

CHAPTER 6

High level design

With ParalleX, HPX, and the security prerequisites covered in chapters 2, 3, and 4 respectively, we can introduce a design to add the access control to HPX.

The design consists of two globally separate parts, the access control and the guarding of the integrity required to support the access control, both of which will be treated below. This will generally be done by weighing the options presented in chapter 4 against the requirements we presented in chapter 5, selecting one.

6.1 Identification

In section 4.1.1 we have seen that identities and identifiers are crucial to the rest of the security mechanisms, thus we must first select the subjects. As the objective is to protect HPX at the component level the components obviously play a vital role, but there are too many in an average application to authenticate them individually using a TTP. Here HPX its hierarchical design can be used to our advantage, with localities at the root of this hierarchy. The component factories are below the localities, which in turn are followed by the components themselves, allowing capabilities to be delegated through this hierarchy.

Each of these, the localities, component factories, and components have a GID as described in section 3.3 which can be used as an identifier. It is is tied irreversibly to a single subject,

unchangeable throughout the lifetime of the subject, and can not be transferred to other subjects.

However, as the locality and component identifiers within the GID are 32-bit and 80-bit respectively there is a chance of a GID being reused. This is only a problem in applications where localities are constantly joining and parting though as the 80-bit limit will practically never be reached. A solution to this problem will be discussed in section 6.3.1.

6.2 Authorisation

In a distributed system both the approaches described in 4.1.3, access control lists and capabilities, have their merits. An access control list approach implements some centralised control and supports administrative activities better. For example, it can easily answer questions such as which subjects have what access to a particular object, which is a commonly asked question when something goes wrong. The ability to answer this is called traceability.

However, checking the validity of a capability is cheaper because it can be done locally, whereas in the access control list approach either an expensive replication or a slow centralised check has to be done. As performance is a primary concern within HPX the use of capabilities over access control lists was obvious, an extra network roundtrip every time an authorisation has to be verified is simply too expensive.

As the extra roundtrip to a centralised server is too much overhead the capabilities are an obvious choice. The most important feature of a capability system is that they can be precomputed and distributed as certificates, which is exactly what we propose to do for this design. An interesting advantage of this is that the system will continue to function even when the authentication server is not available.

Whilst the above advantages are exactly what we are looking for it must be mentioned that a capabilities system can not express specific denial of access rights, which might be useful in some cases. [11]

6.3 Certificate scheme

Three certification schemes were described in section 4.4, a Certificate Authority based approach, an Identity-Based Cryptography approach, and the web of trust. This last option can easily be ruled out though as most HPX applications simply do not run long enough to build such a web. Secondly a new web would have to be built each time an application is started, without any information available on how to bootstrap this trust, thus who to trust initially. Last of all, in this setting the only way to verify whether a

subject is trustworthy is by executing an action on multiple localities and comparing the results, a huge waste of resources.

Whilst the IBC approach has several advantages, there is no need for certificates, subjects do not need to memorise public keys as they can be recomputed, and the KGC can be distributed they do not outweigh the disadvantages. First and foremost, it lacks performance as it still takes several milliseconds to compute a pairing at the 128-bit security level on a modern processor. [33, 10, 8] Secondly, as previously noted, the subject must acquire the key over a secure channel, a channel that is resistant to overhearing and tampering as it is a private key.

With recent progress in the field of elliptic curve cryptography, especially with regard to performance, the CA based approach has become feasible. Whilst it is not perfect, the CA is a single point of failure, it fits the hierarchy described in the previous section nicely as shown in figure 6.1.

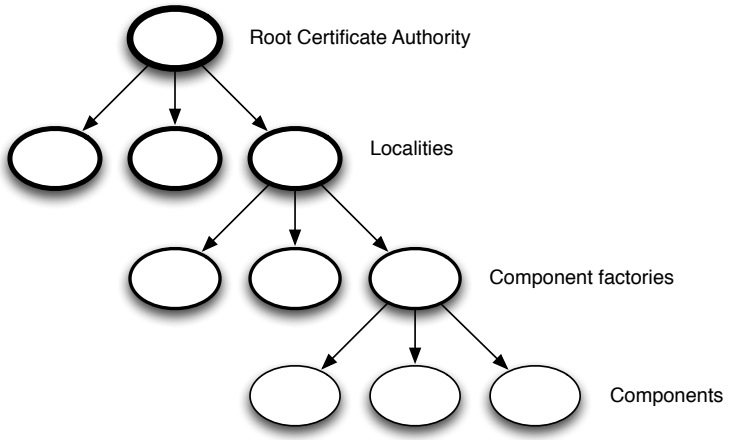


Figure 6.1: The Certificate Authority hierarchy

The CA approach also allows a CORBA [12] inspired multi-levelled design, supporting three levels of security. The first simply does not implement any security features at all, as is the status quo for HPX. At the second and third levels the capabilities are checked on a locality and component level respectively.

6.3.1 Certificates

Now that we have established that we will be using the CA approach we must look at the certificated themselves.

X.509 is an ITU-T standard for a PKI as described in section 4.4.1, and it specifies standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm. The public key certificate is the most widely used part of this standard and has undergone three revisions to date with X.509 version 3 adding a format for certificate extensions. These can be used to optionally store additional information regarding the subject and to define certificate usage.

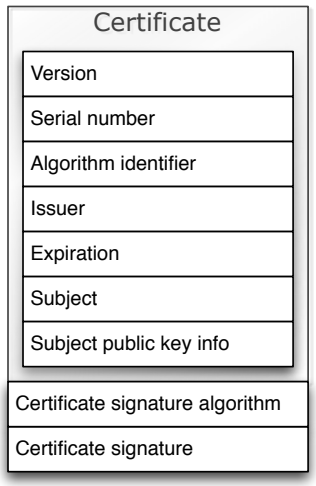
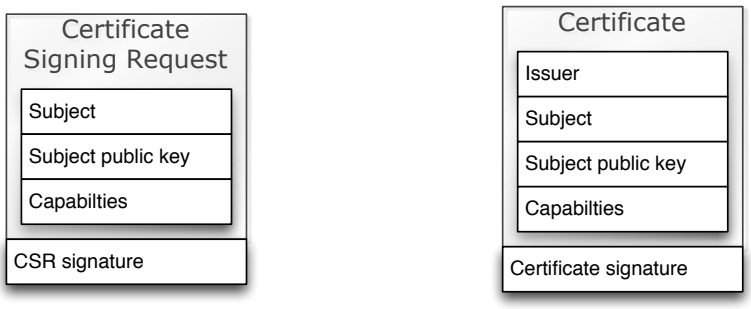


Figure 6.2: The required fields of an X.509 public key certificate

Figure 6.2 lists the required fields of an X.509 certificate and clearly shows how a subject is tied to its public key via the issuer its signature. This standard was designed generically to solve a wide variety of problems, and is as such excessive for our purpose. To keep the size of the certificate signing request and certificate small, minimising the transmission overhead, we have designed our own formats which have noticeably fewer fields as shown in figures 6.3a and 6.3b respectively.

As shown by the extensions in the X.509 version 3 certificates essentially anything can be stored in a certificate, irreversibly linking it to the subject, as long as it can be represented by a series of bytes. In our case we will be using this to store the capabilities of the subject as hinted at in section 6.2. This can be done in any format as long as it meets the aforementioned serialisability requirement.



(a) The certificate signing request format

(b) The certificate format

Figure 6.3: The certificate signing request and certificate formats

The previously mentioned problem of a GID being reused can be solved by optionally adding an expiration date to the certificate, requiring the subjects to renew their certificates periodically. After a subject seizes to exist and its certificate has expired the GID can be reused by a new subject, which in turn requests a new certificate. However, as the chances of this occurring are minuscule and can be detected without the extra field this should only be enabled when the application is expected to run out of GIDs, when run on an extremely dynamic network or large number of nodes. In the latter case it is advisable to increase the size of the locality identifier though as the expiration date adds extra strain on and availability requirements to the CA.

6.3.2 Trust anchor

A trust anchor is an authoritative entity for which trust is assumed and not derived. In this design the root certificate is the trust anchor from which whole chain of trust is derived via the previously described certificate hierarchy. This trust anchor must be in possession of the trusting parties, the localities, before they are able to do any further certificate path validation. To safely distribute the root certificate the root CA must be started before all localities to generate the self-signed certificate, either within or outside of the localities' network, at which point there are two ways for the localities to obtain it.

The first is by distributing it through an out-of-band trusted channel. This may be the user who started the root CA manually providing the certificate to the localities through their command-line, a safe option but one which requires user interaction. In a cluster the filesystem may be a sensible alternative as it is shielded from all other users except the cluster administrator. The root CA could write its certificate there to be picked up by the localities.

A second alternative is for the localities to create the trusted channel themselves. The localities authenticate the root CA after which ephemeral shared secrets are established to safely receive the certificate. This, however, is merely an abstraction as we have seen that authentication requires something to validate against in section 4.1.2.

In summary the solution depends on the trusted channels available, manually providing the certificate via the command-line is the safest option but requires user interaction. Lacking the ability to interact in a cluster another out-of-band channel such as the filesystem might be available, falling back to the locality establishing one itself if none are.

6.4 Authentication

Each time a subject in the hierarchy, thus a locality, component factory, or component, is constructed it generates a new key pair and requests a certificate from the CA immediately above it through a CSR. The CA then has to decide whether to honour the request by verifying the identity of the subject.

This is trivial for the component factories and components as they are constructed by the localities and component factories above them respectively, but this is not the case for the localities requesting a certificate from the root CA. In the localities' case they need to prove their identity to the root CA and this may also decide whether or not the capabilities requested through the CSR are assigned.

The exact method used for a locality to prove their identity will depend on the application and the settings in which it will be used. Within a cluster for example the mere fact that it connects from within the cluster network may be enough proof, whilst in other cases an authentication protocol such as the Extensible Authentication Protocol (EAP), Kerberos, or Secure Remote Password (SRP) [32] may be required.

6.5 Integrity

In sections 4.5.1 and 4.5.2 message authentication codes and signatures are explained respectively. We have noted that symmetric cryptographic algorithms are generally significantly faster than the asymmetric ones thus it seem obvious to protect the communication messages with MACs, especially since non-repudiation is not an objective.

Whilst that solution has been explored by several patents, the latest of which from Sun Microsystems Inc. is US 5,852,666 [22] it does not scale to the number of subjects in an average HPX application. Establishing a shared secret between all communicating pairs of components and their predecessors in the CA hierarchy leads to too much overhead in

two distinct ways, the communication overhead as well as the storage overhead, storing all these secrets adds up.

Though signatures are a little slower they do scale to the required degree. They also allow the balancing of storage and communication overhead as the certificates for localities wishing to communicate may be exchanged when establishing the channel whilst the certificates for the component factories and components may be sent with each message. Several optimisations in this area have been explored in the research of Vehicular Ad Hoc Networks (VANETs) and may also apply here. [16]

6.6 Summary

Figure 6.4 shows how we have extended HPX compared to the figure 3.1 which was first introduced in chapter 3. As mentioned at the beginning of the chapter the design consists of two globally separate parts, the access control and the guarding of the integrity required to support the access control.

The figure clearly shows the certificate hierarchy with a root authority, the intermediate locality and component factory authorities, and the certificate in each component. Secondly it shows the extension to the parcel transport layer to ensure integrity on the network level.

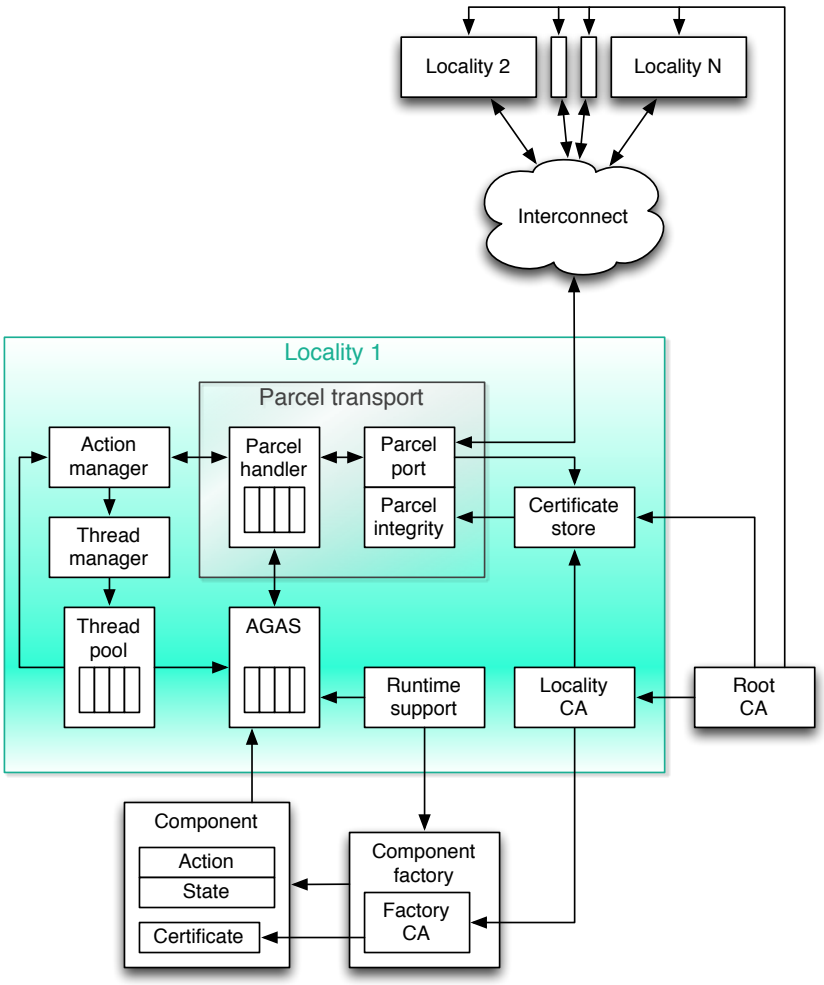


Figure 6.4: The High Performance ParalleX runtime architecture with added features

CHAPTER 7

Implementation

A proof of concept implementation of the high level design presented in the previous chapter has been done up to the locality level and can be found integrated into HPX at `https://github.com/STEllAR-GROUP/hpx`.

As the this proof of concept closely follows the design presented in chapter 6 we have chosen to explore several interesting issues we ran in to whilst developing the implementation in this chapter, and how they were solved.

7.1 Capabilities

As described in section 6.3.1 a certificate can be used to store essentially anything, irreversibly linking it to a subject, as long as it can be represented by a series of bytes. This means in the most trivial form the capabilities may be stored as a bit array, each bit indicating whether or not the subject has a specific capability.

In section 6.3 we have shown that we will be working with a certificate hierarchy, which means we will have to deal with the delegation of capabilities from the root to the intermediate CAs. The most straightforward solution is to only allow an intermediate CA to have capabilities its parent has as well.

This however means the root CA will need all capabilities that its subordinates need, whilst in practice it should not need, or even use them. To solve this we use two bits per capability, the first indicating whether the CA can delegate the capability to its subordinates and the other which indicates whether it has the capability itself. This way the root can delegate capabilities without requiring to have them itself.

Thus a CA with the delegation bit set for a capability C may have a subordinate that has neither the delegation or capability bit, just the delegation bit, just the capability bit, or both. The CA is trusted to make the right decisions. A CA without the delegation bit may only have a subordinate the does not have either the delegation or capability bit.

Whether a certificate is allowed to be used as a CA in the first place is also encoded as a capability. In the implementation the root certificate has this capability set, and all delegation bits.

7.2 libsodium

There is an unwritten rule in cryptography and software engineering that one should never implement their own cryptographic algorithms, there are too many, often subtle, mistakes to be made. However, the design still required this functionality to achieve its goals.

We solved this problem by using libsodium, a portable fork of NaCl [1] which contains an implementation of the Ed25519 public-key signature system [2] as well as several cryptographic hash functions. These include SHA-2 and the newer BLAKE2, an improved version of the SHA-3 finalist BLAKE.

These are the same secure implementations as in NaCl, but packaged in such a way that they are portable, and libsodium does indeed port to all platforms on which HPX can be used, including Android, and more. The library is a great fit as its design goals match that of the high level design and HPX, security, speed, and portability, in that order.

7.3 Certificate store

As this proof-of-concept implementation is only on the locality level the number of certificates that has to be dealt with is significantly lower. The maximum number of localities on which HPX has been tested thus far is 1024. This, combined with the fact that the certificates described in section 6.3.1 are only 144 bytes in size allows each locality to store, in the worst case, a copy of the root certificate and all other localities' certificates.

In section 6.3.2 we explained the root certificate is the trust anchor, thus this is used to initialise the certificate store. Successively, upon adding a locality certificate it is verified,

which means its signature and issued capabilities are checked against the issuer which has to be present in the certificate store already.

Once verified it is stored and considered trusted, which means that next time either a certificate or parcel has to be verified it does not need to be verified against the whole chain. It only needs to be verified against the first item in the chain present in the certificate store, reducing computational overhead.

A second advantage of the certificate store is that the locality certificate only needs to be sent when first connecting to another locality, with one minor exception which will be explained in section 7.4.2, saving network overhead. This still holds when expanding to the component level, only sending the component and component factory certificates with each request keeping the locality certificate stored.

7.4 Parcel suffix

The parcel suffix is a small piece of information as shown in figure 7.1 that is appended to each parcel to allow the receiver to verify its integrity. Each parcel contains a unique identifiers and the parcel suffix is tied to the parcel via this identifier.

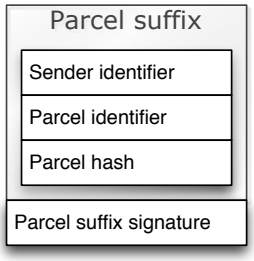


Figure 7.1: The parcel suffix format

The equality of these identifiers is the first thing to be verified after receiving a parcel. Successively the parcel suffix signature is verified using the certificate in the store matching the sender identifier. Last of all the hash value of the parcel is compared to the hash value in the parcel suffix, and when these match the parcel is accepted and processed.

7.4.1 Hashing

When signing data using libsodium we ran into a problem with the library, the API is designed in such a way that it is as hard as possible for a developer to make a mistake.

An important design consideration for a cryptographic library but it does come at a cost, signing a message, or in our design a parcel in our case, copies the parcel. Since parcels may be multiple gigabytes in size this is an expensive operation, both due to a blocking memory allocation as well as actual memory usage itself.

To work around this problem we create a two-stage process, first of all the parcel is hashed using a cryptographic hash function, after which the hash value, together with the rest of the parcel suffix is signed. As such only the parcel suffix needs to be signed, and thus copied, which is small in size compared to the parcels themselves. This provides the same security guarantees as signing the whole message at a fraction of the cost.

7.4.2 Routing

In HPX two localities may need to communicate via a third locality as shown in figure 7.2, where locality 2 sends a parcel to locality 3 via 1. This occurs when a locality is unaware of the final destination its address, either when bootstrapping the runtime or due to an unfortunate cache eviction within AGAS.

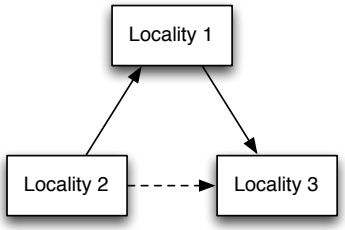
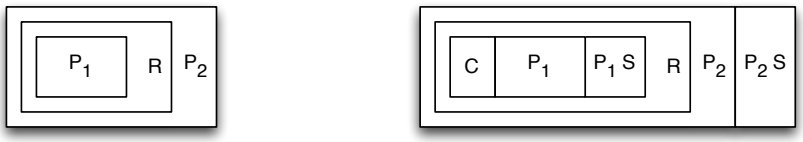


Figure 7.2: Routing a parcel from locality 2 to 3 via 1

To achieve this an intricate encapsulation is used as shown in figure 7.3. First of all the parcel P_1 is encapsulated in a special routing layer R, to indicated to the receiver that it must unpack the parcel and forward it to its destination. This in turn is encapsulated again in a parcel P_2 such that it can be sent through the parcel transport layer.

This is the reason the parcel suffix contains the source identifier, as the signature is not necessarily from the locality which made the connection. To provide security in this edge case as few things need to happen, first of which is prepending the source locality its certificate C to the inner parcel P_1, as we do not know whether the destination locality is already aware of it. The parcel suffix is then appended to P_1 as usual and the parcel is encapsulated in R. Finally this is sent as a parcel complete with parcel suffix to the routing locality which unpacks R and forwards the containing parcel.



(a) A parcel encapsulated to be routed (b) A parcel encapsulated to be securely routed

Figure 7.3: Encapsulation of parcels

CHAPTER 8

Evaluation

In this chapter we will evaluate both the security and performance requirements which were initially formalised in chapter 5, in order to answer the research question regarding viability.

For each of the requirements we will discuss how it influenced, and was incorporated into the design, and into the implementation. Then, where applicable, this will be backed up by an experiment showing whether or not the requirement has been fulfilled.

8.1 Experiments

Unless otherwise specified all experiments in this chapter were done using HPX 0.9.6 which is the first release to contain my contributions and can be found at `https://github.com/STEllAR-GROUP/hpx/releases`. It was compiled using gcc 4.7.3 and the versions where security was enabled were linked against libsodium 0.4.3, which can be obtained from `https://github.com/jedisct1/libsodium/releases`. Additionally boost 1.54.3, hwloc 1.7.0, and tcmalloc 4.1.0 were used.

The experiments were run on the Beowulf partition of the Hermione cluster located at the Louisiana State University. This consists of 16 HP ProLiant DL120 6G nodes, each

containing a quad-core Intel Xeon X3430 at 2.4GHz and 12 gigabytes of memory. These nodes are interconnected by a gigabit ethernet network.

8.2 Security requirements

As explained in section 5.3.1 security is our primary concern, and backed this up by a number of security requirements. These will each be evaluated in the sections below following the design, implementation, and where applicable experiment subsections as explained above.

8.2.1 SR-1

This first requirement states that a locality shall be authenticated before remotely executing actions.

8.2.1.1 Design

The design fulfils this requirement by not allowing a locality to remotely execute an action unless the parcel with which the action is dispatched is signed using a trusted certificate. Thus a certificate which can be verified up to the trust anchor. For a locality to obtain such a certificate it must issue a CSR to the root CA as explained in section 4.4.1, which in turn authenticates the locality before issuing the certificate.

This allows us to start with a root CA, the trust anchor as explained in section 6.3.2, and a locality representing the application. From there we can inductively expand this by authenticating and allowing a locality to remotely execute actions and as such participate in the trusted cluster.

8.2.1.2 Implementation

The implementation follows the design, with the note that the authentication procedure at the time of writing is a stub, thus all CSR requests are honoured with their requested capabilities. It does however allow for two roundtrips as required for authentication protocols such as the SRP protocol, or any of the others described in section 6.4.

8.2.2 SR-2

This next requirement states that access control shall be added to prevent a component from executing actions it is not allowed to.

This encompasses one of the objectives of the design, adding an access control system to the HPX runtime and applications. After allowing an authorised locality to join, as covered by the previous requirement, it shall only be allowed to execute the component actions that is has permissions for.

8.2.2.1 Design

The design expanded upon the requirement by introducing a total of three levels of access control, where the first simply does not implement any access control at all. At the second and third levels the capabilities are checked on the locality and component level respectively.

This is done through the certificate hierarchy described in section 6.3, together with the certificates containing the capabilities as described in sections 6.3.1 and 6.2.

8.2.2.2 Implementation

The implementation is a proof of concept up to the locality level as described in chapter 7, leveraging libsodium and certificate stores to implement a fast version of the certificate hierarchy mentioned above.

There are two problems scaling this implementation to the component level though. As section 3.1 explains HPX maps the notion of a locality to the system-specific concept of a conventional process. Each process has a separate local virtual address space, thus within a process every part of the memory is readable by the code within that process. This means that a component could access, and ultimately use the private keys of other components within a locality, or even delegate rights to new components using the component factories' keys.

Whilst this is not likely to be a problem in practice due to the trust hierarchy, it must still be considered. Even though the components are created through component factories which only do so if they trust the component, these components have access to private data they should not have access to which is a serious violation.

What's more, solving this would require every component to have its own address space, meaning it would need to run as a separate process or aided by the kernel, or hardware support. This, unfortunately, is unviable at this point though it may become viable in

the future as there are several parts within HPX that could benefit from OS or hardware support.

The second problem is that the component libraries are loaded from separate files during application startup. These files contain the components and component factories, and should preferably be signed as an attacker with access to these files could modify them to behave differently. This, however, is difficult as it requires a PKI that has a lifetime far beyond that of a single application instance, and is hence considered outside the scope of this thesis.

8.2.3 SR-3

This security requirement requires the access control mechanism shall be flexible enough to adapt to the application developers' needs.

8.2.3.1 Design

As discussed in section 4.1.3 authorisation consists of two separate processes, assigning a set of authorisations that define what a subject can, and can not do within HPX and ensuring that a subject has sufficient rights to perform the requested operation and preventing those without sufficient rights from doing the same.

The first is covered by the capabilities stored within the certificates that are using in the design, as described in section 6.2. The exact capabilities to be used are left to the application developer, as is the verification as this depends on the capabilities chosen.

8.2.3.2 Implementation

In section 7.1 we have described our implementation and the reasoning behind it. By default four different capabilities are in place as well as bits indication whether delegation of these capabilities are allowed.

These are the capability to create a subordinate CA, to remotely construct a component, to execute a const component action and finally one to indicate whether it may execute a non-const component action.

8.2.4 SR-4

This requirement states that a connecting locality shall not impede the application its availability.

8.2.4.1 Design

As explained by the CIA triad in chapter 4 there is a delicate balance between confidentiality, integrity, and availability. The high level design presented in chapter 6 adds access control to prevent unauthenticated localities remotely executing actions and prevent both localities as well as components from executing actions they do not have the capabilities to access.

Once authenticated and in possession of the required capabilities it does not impede a locality from working together with the rest of the cluster to execute an application. This is of course within the rights of the obtained capabilities, thus allowing access to all data and functionality within the rights of the capability.

8.2.5 SR-5

This last security requirements states that parcels' integrity shall be guarded when sent over an untrusted network.

8.2.5.1 Design

In chapter 4 we mentioned that there are essentially two types of mechanisms when it comes to safeguarding integrity, the preventive and detective mechanisms. The design detailed in section 6.5 uses the techniques described in section 4.5 in a detective manner as we do not control the network and as such the preventive mechanisms are simply not an option.

8.2.5.2 Implementation

The implementation is described in detail in section 7.4 and follows the design referenced above.

This is only the first half of the detection mechanism though, and begs the question of how to handle a detected modification. Here it is important to note that the underlying network protocol, TCP/IP, also includes integrity handling, thus if the detection is triggered at the parcel transport layer it is either because the parcel was incorrectly constructed at the source locality or due to intentional modification by an attacker.

There are essentially two ways of handling such a modification, requesting the source locality to resend the initially modified parcel or removing the source locality from the application, which could be combined in a policy where the locality is removed after a number of modified transmissions for example.

The resending of parcels is expensive however, as it means the locality must temporarily store them until they have been received in good order. Secondly, since HPX has not yet implemented the migration described in section 2.4 we are not able to forcefully remove a locality from an application without giving up on the application as a whole. As such the current implementation forcefully terminates the application after logging an error. This fail-fast approach is safer than continuing in a compromised environment, and allows the cause of the discrepancy to be solved faster.

8.3 Performance requirements

Although a second priority after security, the performance of the design is still crucial to its success. The performance requirements listed in section 5.3.2 will be evaluated below in a similar manner as the security requirements above were.

8.3.1 PR-1

The first performance requirement states that no more than 250 microseconds of overhead shall be added in dispatching a parcel.

8.3.1.1 Design

The high level design takes performance into account at every turn, choosing the more performant options where possible. This is not always possible though as we have seen in section 6.5 for example, where we were forced to choose a less performant option because of scalability reasons.

8.3.1.2 Implementation

The proof of concept implementation follows the design to show it to be functional, but has as such not been extensively optimised yet.

8.3.1.3 Experiment

HPX has a separate collection of benchmarks including an implementation of the Ohio State University (OSU) latency benchmark originally developed for MPI, which can be found at `https://github.com/STEllAR-GROUP/hpx_benchmarks`.

This latency test is carried out in a ping-pong fashion where the sender sends a parcel of a certain size to the receiver and waits for the receiver to reply with a parcel of the same size. Many iterations of this ping-pong test are carried out and average one-way latency numbers are obtained per size.

Figure 8.1 shows this benchmark being carried out with three different configurations of HPX, the first of which has security disabled whilst the other two have security enabled but with different hash functions, SHA-2 and BLAKE2 respectively.

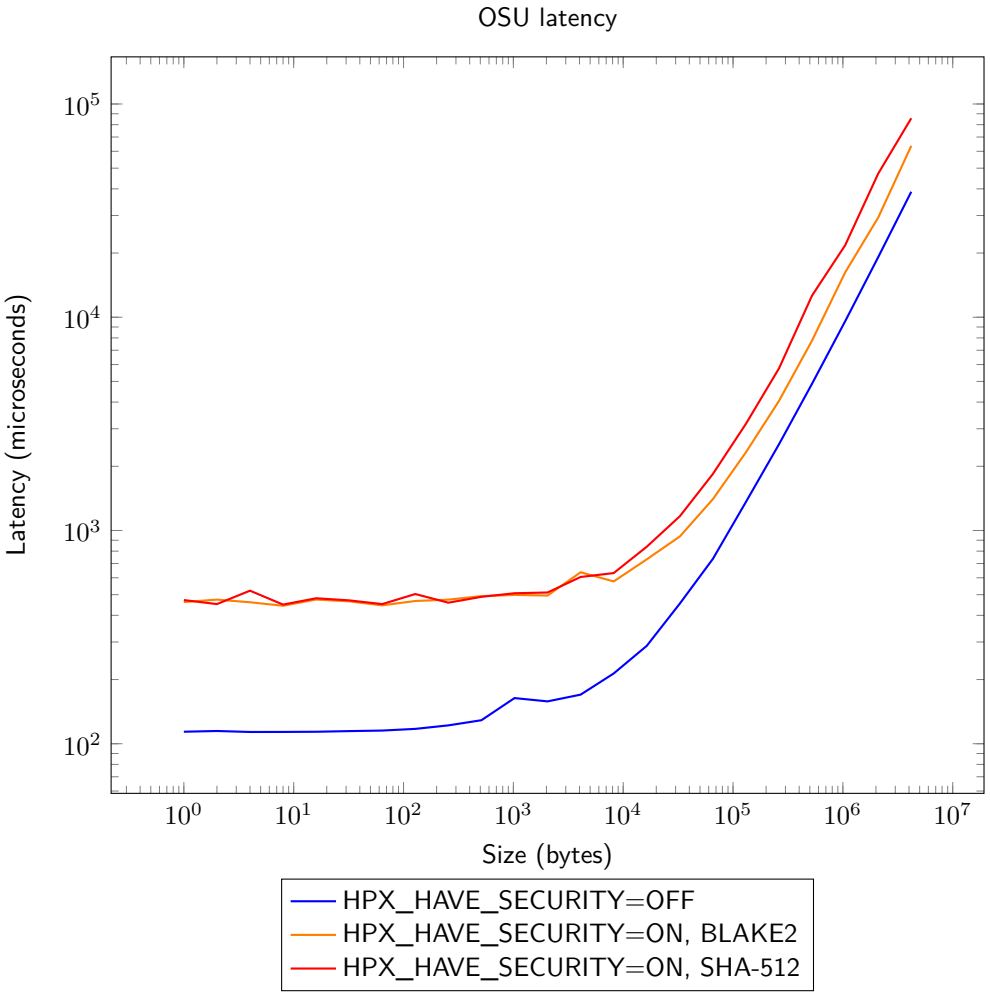


Figure 8.1: Ohio State University latency

Here we can see that initially, with parcels under 2048 bytes, there is around 348 microseconds of added overhead when security is enabled compared to when it is disabled. This can be fully contributed to the signing of the parcel suffix, about 100 microseconds on the sending side and 250 microseconds on the receiving side.

HPX contains performance counters through which these values were obtained, counters which expose critical information about different modules of the runtime system. They are designed to help determine system bottlenecks and fine-tune system, and application performance. Listing 1 lists the output of an example application its counters obtained from HPX via `--hpx:print-counter /security/time/tcpip/received` et cetera.

```
/security{locality#0/total}/time/tcpip/received,1,14.663712,[s],3.11768e+07,[ns]
/security{locality#1/total}/time/tcpip/received,1,14.667476,[s],2.69671e+07,[ns]
/security{locality#0/total}/time/tcpip/sent,1,14.675712,[s],1.20604e+07,[ns]
/security{locality#1/total}/time/tcpip/sent,1,14.655537,[s],1.2276e+07,[ns]
/parcels{locality#0/total}/count/tcpip/received,1,14.675719,[s],124
/parcels{locality#1/total}/count/tcpip/received,1,14.667871,[s],124
/parcels{locality#0/total}/count/tcpip/sent,1,14.663740,[s],122
/parcels{locality#1/total}/count/tcpip/sent,1,14.667914,[s],126
```

Listing 1: Example performance counter

As the parcels grow over 2048 bytes however the overhead created by the signing of the parcels slowly becomes negligible and we see the latencies grow. In the configuration where security is disabled this is primarily due to parcels having to be split over multiple TCP packets, as the Maximum Transmission Unit (MTU) of the network is 1500.

The growth is steeper with security enabled which is caused by the hashing of the parcel, as can be seen by the difference between the SHA-2 and BLAKE2 algorithms. Here the growth starts around the 2048 byte size mark as at that point the parcel no longer fits in the CPU cache, which means that main memory needs to be accessed during hashing.

If we map the latency increase of the configurations with security enabled compared to the version with security disabled as in figure 8.2 we see that the relative overhead decreases with the parcel size. This matches the behaviour of TCP where sending a single large parcel is more efficient than sending several smaller parcels. To use this optimisation HPX has implemented parcel coalescing which combines several small parcels sent within a short time frame into a single parcel.

Whilst this is outside of the 250 microseconds that we had set as a requirement this can still be considered acceptable. Although there is a noticeable amount of overhead it is not enough to render HPX unusable, especially when using the BLAKE2 hash function.

Figure 8.2: Ohio State University latency increase

8.3.2 PR-2

This last requirements states that no more than 5 percent overhead shall be added to an application by the access control.

8.3.2.1 Implementation

In section 8.3.1.2 we explained that the proof of concept implementation was to show the high level design to be functional. This requirement shows, through the experiment below, that it is by running an example application.

8.3.2.2 Experiment

This experiment is based on the fibonacci_futures_distributed example within HPX, which, as the name implies, can be used to calculate a fibonacci number. It does this using an algorithm with a computational complexity of $O(2^n)$ however, because it is representative for a whole class of applications.

The algorithm represents tree based recursive data structures such as those used in game theory and Adaptive Mesh Refinement (AMR), the latter of which is an important method for a wide range of physics simulations. It is also representative of graph based algorithms such as breadth first search since it is characterised by very tightly coupled data dependencies between calculations.

The fibonacci_futures_distributed example accepts three parameters, the first of which is `--n-value` indicating the fibonacci number to calculate. Next there is `--threshold`, which is the threshold for switching to a serial implementation in order for each thread within HPX to have enough work. Furthermore there is `--distribute-at` specifying at which point it moves the sub-tree to a locality, allowing the amount of work all localities have to perform to be varied.

Figure 8.3 shows the runtime of the application when run with `--n-value` 48 `--threshold` 28, varying the `--distribute-at` value. Here we can see the performance cost that enabling security has, a factor 1.55 increase in runtime on average. Whilst having more work per locality, by varying the `--distribute-at`, generally allows the latencies to be better hidden this did not influence the runtimes in this case.

As the parcels are small in size there is no significant difference between the runs using the SHA-2 hash function verses those using BLAKE2.

Unfortunately this result is far outside of the required 5 percent, though not enough to render it unusable when the security is required. When it is required it needs to be considered carefully though, as it does come at the expense of performance.

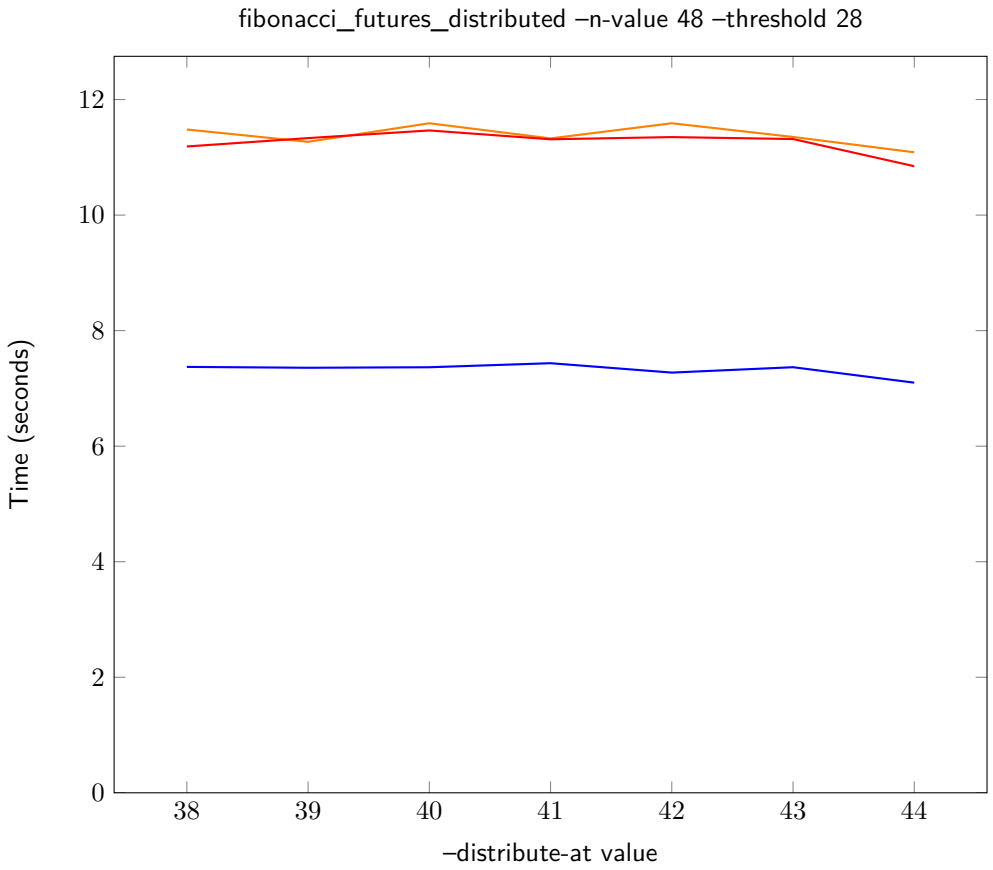


Figure 8.3: fibonacci_futures_distributed on two localities

CHAPTER 9

Conclusions

In this thesis we have designed and implemented a security solution for HPC leveraging the features of the ParalleX execution model, and its implementation HPX. It adds access control by requiring localities joining a simulation to authenticate to a root certificate authority, receiving a certificate containing a set of capabilities describing their rights. The design also guards the integrity of parcels using signatures to detect unauthorised modifications.

The first priority within the design was security, and this is viable up to the locality level. The core of the design presented in chapter 6 held up to the evaluation presented in chapter 8, with the notable exception of the inner-locality memory access which theoretically allows components to access other components' private keys.

As explained in section 8.2.2.1 the only solution to this would be to run each component within its own local virtual address space as controlled by the OS, potentially aided by the kernel or hardware support. Without such support this is unviable though, as it would mean running each component within its own process.

A second priority was minimising the performance overhead associated with the security mechanisms. Some added overhead was unavoidable as shown by the performance evaluation in chapter 8. These showed an increase in communication latencies, one of the four limiting factors as in the *SLOW* acronym introduced in chapter 2, by a factor four through

1.6 depending on the parcel size. Thus while feasible, the solution has a significant cost and will likely only be acceptable for applications with high security demands.

Whilst the high level design has taken performance into account at every turn the proof of concept implementation used in the evaluation was primarily to show the design to be functional, and has not been extensively optimised. As such we are optimistic that further optimisations can improve the implementation its performance.

The current implementation enables, or disables these security features globally within an HPX application though they could also be enabled on a per locality basis. In such a scenario localities within a cluster would be deemed trusted and be allowed to run without integrity guards whereas the localities connecting from the outside, over an unsecured network, would be protected. This provides further flexibility in the balance between security and performance.

It must also be mentioned that the implementation has been done in such a way that it is portable to all platforms that HPX supports. This is primarily due to the excellent libsodium library as described in section 7.2.

Ultimately, when and where to enable the provided security features needs to be considered carefully as they do come at the expense of performance. The performance evaluation in chapter 8 support the user in making a suitable trade-off between security and performance.

9.1 Future work

In addition to the optimisation of the proof of concept implementation as mentioned above there are several interesting problems that were not explored. This was either due to time constraints or the fact that they were clearly outside the scope of this thesis.

9.1.1 Integrity

In section 6.5 we chose to solve the integrity guarding using signatures, as a solution based on message authentication codes did not scale to the number of components in an average HPX application. However, this solution likely does scale to the number of localities when security is not required to the component level, which may be an interesting design optimisation.

The section also mentioned research in the area of VANETs that might apply, at least partially, to this design which could be explored.

9.1.2 Encryption

In the design we assumed, based on the scenarios and goals in chapter 5 that the parcel content itself was not confidential. This could be the case in the medical field though, for example, and as such the design could be extended to optional encrypt this content. This will undoubtedly have a further impact on performance, though it is to be seen how much and whether the approach would still be viable.

Bibliography

[1] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. "The security impact of a new cryptographic library". In: *Proceedings of the 2nd international conference on Cryptology and Information Security in Latin America*. LATINCRYPT'12. Santiago, Chile: Springer-Verlag, 2012, pp. 159–176. ISBN: 978-3-642-33480-1. DOI: 10.1007/978-3-642-33481-8_9. URL: http://dx.doi.org/10.1007/978-3-642-33481-8_9.

[2] Daniel J. Bernstein et al. "High-speed high-security signatures". In: *Proceedings of the 13th international conference on Cryptographic hardware and embedded systems*. CHES'11. Nara, Japan: Springer-Verlag, 2011, pp. 124–142. ISBN: 978-3-642-23950-2. URL: http://dl.acm.org/citation.cfm?id=2044928.2044940.

[3] Francois Cantonnet et al. "Fast Address Translation Techniques for Distributed Shared Memory Compilers". In: *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers - Volume 01*. IPDPS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 52.2–. ISBN: 0-7695-2312-9. DOI: 10.1109/IPDPS.2005.219. URL: http://dx.doi.org/10.1109/IPDPS.2005.219.

[4] B.L. Chamberlain, D. Callahan, and H.P. Zima. "Parallel Programmability and the Chapel Language". In: *Int. J. High Perform. Comput. Appl.* 21.3 (Aug. 2007), pp. 291–312. ISSN: 1094-3420. DOI: 10.1177/1094342007078442. URL: http://dx.doi.org/10.1177/1094342007078442.

[5] Philippe Charles et al. "X10: an object-oriented approach to non-uniform cluster computing". In: *SIGPLAN Not.* 40.10 (Oct. 2005), pp. 519–538. ISSN: 0362-1340. DOI: 10.1145/1103845.1094852. URL: http://doi.acm.org/10.1145/1103845.1094852.

[6] Chirag Dekate et al. "Improving the scalability of parallel N-body applications with an event-driven constraint-based execution model". In: *Int. J. High Perform. Comput. Appl.* 26.3 (Aug. 2012), pp. 319–332. ISSN: 1094-3420. DOI: 10.1177/1094342012440585. URL: http://dx.doi.org/10.1177/1094342012440585.

[7] W. Diffie and M. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theor.* 22.6 (Sept. 2006), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638. URL: http://dx.doi.org/10.1109/TIT.1976.1055638.

[8] Nicolas Estibals. "Compact hardware for computing the tate pairing over 128-bit-security supersingular curves". In: *Proceedings of the 4th international conference on Pairing-based cryptography*. Pairing'10. Yamanaka Hot Spring, Japan: Springer-Verlag, 2010, pp. 397–416. ISBN: 3-642-17454-X, 978-3-642-17454-4. URL: http://dl.acm.org/citation.cfm?id=1948966.1949002.

[9] Donald L. Evans, Phillip J. Bond, and Arden L. Bement. *Standards for Security Categorization of Federal Information and Information Systems*. Feb. 2004. URL: http://csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf.

[10] Mohammad Sabzinejad Farash et al. "A new efficient authenticated multiple-key exchange protocol from bilinear pairings". In: *Comput. Electr. Eng.* 39.2 (Feb. 2013), pp. 530–541. ISSN: 0045-7906. DOI: 10.1016/j.compeleceng.2012.09.004. URL: http://dx.doi.org/10.1016/j.compeleceng.2012.09.004.

[11] Li Gong. "A secure identity-based capability system". In: *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*. 1989, pp. 56–63. DOI: 10.1109/SECPRI.1989.36277.

[12] Object Management Group. *The Common Object Request Broker: Architecture and Specification (CORBA 2.3.1 specification)*. Tech. rep. Object Management Group, Oct. 1999. URL: http://cgi.omg.org/cgi-bin/doc?formal/99-10-07.

[13] T. Heller, H. Kaiser, and K. Iglberger. "Application of the ParalleX execution model to stencil-based problems". In: *Comput. Sci.* 28.2-3 (May 2013), pp. 253–261. ISSN: 1865-2034. DOI: 10.1007/s00450-012-0217-1. URL: http://dx.doi.org/10.1007/s00450-012-0217-1.

[14] H. Kaiser, M. Brodowicz, and T. Sterling. "ParalleX An Advanced Parallel Execution Model for Scaling-Impaired Applications". In: *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*. 2009, pp. 394–401. DOI: 10.1109/ICPPW.2009.14.

[15] Frank Kargl, Stefan Schlott, and Michael Weber. "Identification in Ad Hoc Networks". In: *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 09*. HICSS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 233.3–. ISBN: 0-7695-2507-5. DOI: 10.1109/HICSS.2006.208. URL: http://dx.doi.org/10.1109/HICSS.2006.208.

[16] Frank Kargl et al. "Secure and efficient beaconing for vehicular networks". In: *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking*. VANET '08. San Francisco, California, USA: ACM, 2008, pp. 82–83. ISBN: 978-1-60558-191-0. DOI: 10.1145/1410043.1410060. URL: http://doi.acm.org.janus.libr.tue.nl/10.1145/1410043.1410060.

[17] Tevfik Kosar and Mehmet Balman. "A new paradigm: Data-aware scheduling in grid computing". In: *Future Generation Computer Systems* 25.4 (2009), pp. 406 – 413. ISSN: 0167-739X. DOI: http://dx.doi.org/10.1016/j.future.2008.09.006. URL: http://www.sciencedirect.com/science/article/pii/S0167739X08001520.

[18] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. United States, 1997.

[19] Butler W. Lampson. "Protection". In: *SIGOPS Oper. Syst. Rev.* 8.1 (Jan. 1974), pp. 18–24. ISSN: 0163-5980. DOI: 10.1145/775265.775268. URL: http://doi.acm.org/10.1145/775265.775268.

[20] Carl E. Landwehr. "Formal Models for Computer Security". In: *ACM Comput. Surv.* 13.3 (Sept. 1981), pp. 247–278. ISSN: 0360-0300. DOI: 10.1145/356850.356852. URL: http://doi.acm.org/10.1145/356850.356852.

[21] Gavin Lowe. "An attack on the Needham-Schroeder public-key authentication protocol". In: *Inf. Process. Lett.* 56.3 (Nov. 1995), pp. 131–133. ISSN: 0020-0190. DOI: 10.1016/0020-0190(95)00144-2. URL: http://dx.doi.org/10.1016/0020-0190(95)00144-2.

[22] Los Altos CA Mark S. Miller et al. "Capability security for distributed object systems". Patent US 5852666 (US). Dec. 1998. URL: http://www.patentlens.net/patentlens/patent/US_5852666/en/.

[23] David McBride. "Building a Better Grid Authentication System with Kerberos". PhD thesis. Imperial College of Science, Technology and Medicine, July 2011. URL: http://pubs.doc.ic.ac.uk/dwm-phd-thesis/.

[24] Maged M. Michael, Martin T. Vechev, and Vijay A. Saraswat. "Idempotent work stealing". In: *SIGPLAN Not.* 44.4 (Feb. 2009), pp. 45–54. ISSN: 0362-1340. DOI: 10.1145/1594835.1504186. URL: http://doi.acm.org/10.1145/1594835.1504186.

[25] R M Needham and M D Schroeder. "Authentication revisited". In: *SIGOPS Oper. Syst. Rev.* 21.1 (Jan. 1987), pp. 7–7. ISSN: 0163-5980. DOI: 10.1145/24592.24593. URL: http://doi.acm.org/10.1145/24592.24593.

[26] Robert W. Numrich and John Reid. "Co-arrays in the next Fortran Standard". In: *SIGPLAN Fortran Forum* 24.2 (Aug. 2005), pp. 4–17. ISSN: 1061-7264. DOI: 10.1145/1080399.1080400. URL: http://doi.acm.org/10.1145/1080399.1080400.

[27] Ruud van der Pas. *Memory Hierarchy in Cache-Based Systems*. Tech. rep. 817-0742-10. Sant a Clara, California, U.S.A.: Sun Microsystems, Nov. 2002.

[28] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[29] Science and Technology Facilities Council. *High Performance Computing explained*. July 2012. URL: http://networking.stemnet.org.uk/sites/default/files/HPC_explained.pdf.

[30] Adi Shamir. "Identity-based cryptosystems and signature schemes". In: *Proceedings of CRYPTO 84 on Advances in cryptology*. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985, pp. 47–53. ISBN: 0-387-15658-5. URL: http://dl.acm.org/citation.cfm?id=19478.19483.

[31] Thomas Sterling. *Towards an Execution Model for HPC Clouds*. Keynote Address at 2012 International Symposium on Grids and Clouds (ISGC12). Taipei, Taiwan, 2012.

[32] Thomas Wu. "The secure remote password protocol". In: *In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*. 1998, pp. 97–111.

[33] Gavin Xiaoxu Yao et al. "Faster pairing coprocessor architecture". In: *Proceedings of the 5th international conference on Pairing-Based Cryptography*. Pairing'12. Cologne, Germany: Springer-Verlag, 2013, pp. 160–176. ISBN: 978-3-642-36333-7. DOI: 10.1007/978-3-642-36334-4_10. URL: http://dx.doi.org/10.1007/978-3-642-36334-4_10.