ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

SEMANTIC VERIFICATION OF SECURITY PROTOCOLS

Graduation Project

Sabri Özgür
150130004

Department: Computer Engineering
Division: Computer Engineering

Advisor: Lect. Dr. Mehmet Tahir Sandıkkaya

June 2017

Özgünlük Bildirisi

1. Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini,
2. Alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılım/donanımın tarafımdan yapıldığını bildiririm.

İstanbul, 29.05.17

Sabri Özgür

Acknowledgments

I thank my family for supporting me throughout my whole education and my advisor Mehmet Tahir Sandıkkaya for guiding and helping me in this project.

Semantic Verification of Security Protocols

(Summary)

Internet is just a communication channel by means of security protocols where servers and clients share information with each other. Security is not included by default in this channel. For this purpose security protocols are implemented and included to communication interfaces to provide secure transmission, authentication and authorization. The goal of the project is to verify whether this protocols are secure or not by modeling different protocols in automatic verification tool called Tamarin Prover. The success of the project should be measured through whether the implemented protocols give valid results or not.

In this project, 4 different protocols are modelled in Tamarin Prover according to the Alice-Bob notation taken from AVISPA website library. These are CRAM-MD5, CHAPv2, APOP, Kerberos. And a Isawa-Morii protocol that is taken from their paper about the one time pads and a way to use them efficiently and securely on internet communications.

Apart from mathematical proof, security protocols are formally proven either computationally or semantically. Tamarin Prover is a semantic prover that does not search the whole solution space computationally, instead it searches using heuristics.

CRAM-MD5 is tested against 3 lemmas. These are shared key secrecy, possible key setup and strong authentication lemmas. All of the lemmas are proven with Tamarin Prover and it is verified that protocol is secure to use.

CHAPv2 is verified with 4 different lemmas: Strong authentication on N_a and N_b values, secrecy of the shared key and possible session setup lemmas. All of them resulted in true and protocol could be counted secure for internet tranmissions.

APOP is verified with 3 different lemmas and these are strong authentication on timestamp and shared key, secrecy of shared key and possible session setup lemmas. All of the outcomes state that APOP is a secure protocol.

Kerberos is tested with 3 lemmas: Secrecy of session key_{CG}, weak authentication on key_{CG} and possible key setup lemmas. Only the key setup lemma is proven, other two lemmas could not be proven because Tamarin Prover finds insecure cases for those lemmas. This may be due to the implementation of the protocol in Tamarin Prover or flaws in the protocol.

One Time Password Scheme is tested with 2 lemmas and these are possible registration and authentication lemmas. Both of them points that the protocol is secure to use. Only these lemmas are used because registration phase happens through a secure channel therefore their security is assumed to be uncompromised.

Güvenlik Protokollerinin Anlamsal Doğrulanması

(Özet)

İnternet güvenlik protokolleri açısından bakıldığında son kullanıcılar arasında bir iletişim kanalı oluşturmak için kullanılır. Bu kanalda iletişim sırasında güvenlik kendiliğinden bulunmaz. Kanal sadece çift yönlü veri taşıma için kullanılır. Bu veriler, konuşmaya dahil olan bireylerin rızası olmadan, kanal dinleme yoluyla rahatça ele geçirilebilir. Bu sebeple kriptografik protokoller ile veriler farklı yollarla şifrelenebilir. Başkaları bu veriye ulaşsa bile şifreli veriden (eğer güvenli bir şifreleme yöntemi kullanıldıysa) anlamlı mesajlar çıkaramazlar ya da çıkarsalar dahi, çıkardıklarının gerçek konuşma olup olmadığından emin olamazlar. Kriptografik protokoller gizlilik, kanıtlama, tanıtma, yetkilendirme, doğruluk gibi farklı güvenlik özelliklerini sağlamak amacıyla oluşturulup internetteki iletişim kanallarına eklenirler. Protokoller farklı son kullanıcılar arasında gönderilmesi gereken mesajları ifade eder. Eğer protokolün çalışması sırasında mesajlar formata uymazsa ya da gönderilen ve alınan verilerin uyuşmadığı fark edilirse oturum sonlandırılır. Güvenlik protokolleri belirlediği mesajlarda farklı matematiksel fonksiyonlar kullanarak bilgilerin güvenliğini sağlar. Bu bağlamda güvenlik kavramı, protokolün sağlamayı hedeflediği güvenlik özelliklerini sağlayıp sağlamaması ile değerlendirilebilir. Bazen şartları sağlasa dahi güvenliğinde noksanlık bulunabilir. Bunun sebebi matematik fonksiyonlarındaki açıklardan kaynaklanmaktadır. Matematik fonksiyonlarının güvenlik protokollerinde kullanılabilmesi için koşulsuz güvenli ya da hesaplama açısından güvenli olması gerekir. Koşulsuz güvenlik pek sık rastlanan bir durum değildir. En az mesaj uzunluğu kadar uzun bir tek kullanımlık şifre ile şifrelenen bir ileti, içeriği hakkında bir bilgi barındırmaz. Farklı şifrelerle farklı anlamlı mesajlar üretilebileceği için, saldırgan iletiye ulaşsa bile, ulaştığının gerçekten gönderilmek istenen metin olduğuna dair herhangi bir çıkarımda bulunamaz. Bu durum koşulsuz güvenliği ifade eder. Hesaplama açısından güvenli kavramı ise çok daha basit bir kavramdır ve genel olarak güvenlik protokollerinde kullanılan şifreleme yöntemleri de bu başlık altında yer alırlar. Hesaplama açısından güvenlik günümüz bilgisayarlarının hesaplama yetenekleri değerlendirilerek ortaya atılan bir kavramdır. Şifreli bir metin verildiğinde bunun tüm olasılıklar denenerek deşifre edilmesi için gereken sürenin dünyada varolan tüm işlem gücü kullanılsa dahi asırlar sürmesi gerekir. Bu şartı sağladığı takdirde bir fonksiyon hesaplama açısından güvenli kabul edilir. Bu projede fonksiyonların güvenlikleriyle, değil güvenlik protokollerinin genel yapısının güvenli olmasıyla ve protokollerin doğrulanmasıyla ilgileneceğiz.

Güvenlik protokollerinin doğrulanması isimden de anlaşılacağız üzere protokolde kullanılan matematik fonksiyonlarının güvenliğiyle değil, protokolün genel akışının güvenliğiyle ilgilenir. Alakalı matematik fonksiyonlar güvenli kabul edilir. Protokoller genel olarak vadettiği özellikleri yerine getirse de, bazı durumlarda, saldırganların içerideki bilgiyi değiştirmesine olanak sağlayan açıklar içerebilir. Bu yüzden doğrulanmamış veya açık bulunduran güvenlik protokollerinin kullanımı, aşağı yukarı güvenlik katmanının tamamen kaldırılması anlamına gelebilir. Bu projede amaç, farklı protokolleri modelleyerek bunlardaki açıkları (varsa) bulmak, olmadığı durumlarda da güvenli olduğunu doğrulamaktır.

Peki, bu doğrulama işlemi nasıl yapılır? Doğrulama işlemi herhangi bir araç kullanmadan insanlar aracılığıyla da yapılabilir. Bu durumda insanlar bilgisayarlarda ifade

edilmesi zor durumları hızlıca tespit edip müdahale edebilir ancak sistematik bir yol izlenmediği için gözden kaçırılan binlerce durum olacaktır. Bunun için güvenlik protokollerinin doğrulanması otomatize edilmelidir ve doğrulama bilgisayar destekli araçlar ile sağlanmalıdır.

AVISPA, otomatik protokol doğrulama araçlarının başta gelenlerinden birisidir. AVISPA projesinin varoluş amacı internet protokollerinin doğrulanmasını dolayısıyla geliştirilmesini hızlandırmaktır. İlk olarak 2001 yılında ortaya çıkıp, güvenlik protokollerinin otomatik doğrulanması konusuna dikkat çekip, destek almayı başarmıştır. AVISPA hesaplama ağırlıklı güvenlik protokol modelleme aracıdır. Verilen protokol için tüm olası durumları hesaplayarak protokolü doğrulamaya çalışır. AVISPA, protokol modellemesi sırasında, her kullanıcı için ayrı bir durum makinası yazılmasını gerektirir. Durum makinalarını birbirilerine karşı çalıştırarak farklı mantıksal önermelerin sağlanıp sağlanmadığını kontrol eder.

Tamarin Prover da 2012 yılında doktora tezi olarak ortaya çıkmış, otomatik protokol modelleme ve doğrulama araçlarından birisidir. AVISPA'ya göre nispeten daha yeni olan Tamarin, gelişmiş görsel arayüzü ile doğrulanmak istenen önermeleri adım adım doğrulama fırsatı sağladığı için AVISPA'ya göre daha hızlı bir protokol geliştirme süreci sunar. Bunun yanısıra Tamarin'deki yapı, 2 farklı durum makinası şeklinde olmaktan öte, birbirini takip eden kurallar olarak ifade edilmiştir. Önermelerde verilen sembollerin olduğu kurallardan başlayarak geriye doğru önceki kuralları tarayarak protokolün doğrulanmasını sağlar. Bu şekilde tüm durumları tarama gereği ortadan kalkmıştır ancak, bunun kötü yanı, aynı güvenlik özelliğini kontrol eden önermelerin farklı protokoller için farklı şekilde yazılmasını gerektiriyor olmasıdır. Bu projede protokoller Tamarin Prover aracında modellenmiş ve doğrulanmıştır.

Matematiksel kanıtlamadan ayrı olarak güvenlik protokolleri ya hesapsal açıdan ya da anlamsal açıdan doğrulanırlar. Tamarin Prover anlamsal bir modelleme dili olup tüm durum uzayını hesapsal açıdan araştırmak yerine anlamsal olarak sezgisel yönlendirmeler ile araştırır. AVISPA ise tüm durum uzayını hesapsal olarak araştırır.

Uzun bir sürecin ardından proje kapsamında AVISPA'da modellenmiş 4 farklı güvenlik protokolü ile 1 tane de makale üzerinde belirtilmiş tek kullanımlık şifre mantığı üzerine kurulmuş ayrı bir protokol Tamarin Prover üzerinde doğrulanmıştır.

CRAM-MD5 protokolü, IMAP4 protokolüne kanıtlama özelliğini ekleyen protokoldür. İstemci, sunucuya kim olduğunu söyledikten sonra, sunucu, istemciye bir test gönderir. Bu testte istemci ortak şifreleri ile testteki değeri birleştirip özünü alarak karşıya göndermek zorundadır. Sunucu aynı işlemi kendisi yapar ve kendi hesapladığı ile aldığı değerleri karşılaştırır. Eğer değerler eşleşiyorlarsa, sunucu, istemcinin iddia ettiği kişi olduğunu kabul eder. Bu protokolün doğrulanmasında 3 farklı önerme kullanıldı. Bunlar oturum oluşturulmasının mümkün olup olmadığını kontrol eden önerme, güçlü kanıtlama önermesi ve ortak şifrenin gizliliği önermesidir. Hepsi doğrulandı ve protokolün güvenli olduğu sonucuna ulaşıldı.

APOP, elektronik posta almak için kullanılan POP protokolünün, kanıtlama işlemini yerine getirmesi için oluşturulmuş bir protokoldür. POP, şifreleri internet üzerinden açık bir şekilde gönderdiği için, güvenlik katmanı hiç bulunmayan bir protokoldür. APOP içinde ortak şifre ve zaman değeri bulunduran bir öz ile kanıtlama işlemini

yaptığı için, güvenli bir protokoldür. Bu güvenliğinin doğrulanması için 3 farklı önermeye karşı test edildi. Bunlar olası oturum, ortak şifrenin gizliliği ve protokol işlemleri sırasında gönderilen test değişkeninin güçlü kanıtlanması önermeleridir. Hepsi doğrulandı ve protokolün güvenli olduğu sonucuna varıldı.

CHAP, şifreleri açık olarak gönderen PAP'ın geliştirilmesi ile ortaya çıkmış bir protokoldür. Microsof CHAP protokolünü, kendi istemcileri ve sunucuları arasındaki bağlantıyı sağlamak amacıyla MS-CHAPv1 ve MS-CHAPv2 olmak üzere iki defa geliştirmiştir. İlkinde tek taraflı kanıtlama bulunurken, ikincisinde hem sunucu hem de istemci karşı taraftan kendisini kanıtlamasını beklemektedir. MS-CHAPv2 protokolünün doğrulanması için 4 farklı önerme yazıldı. Bunların iki tanesi iki ucun birbirini kanıtlamaları için gönderdikleri değişkenlerin önermeleri, bir tanesi olası oturum kurulumu önermesi ve sonuncusu da ortak şifrenin gizliliğini kontrol eden önermedir. Hepsi doğrulanmış olup protokolün güvenli olduğu sonucuna varıldı.

Kerberos, MIT'nin kampüs içi okul sistemlerine bağlantıları düzenleyen protokol olarak ortaya çıktı. İlk 3 sürümü sadece MIT içerisinde kullanılmış olup, Kerberos v4'ün ise, eski şifreleme yöntemlerini kullandığı ve güvenlik açıklarına sahip olduğu için geliştirilmesi durdu. En son sürümü olan Kerberos v5, hala kullanılmaktadır ve Kerberos v5'in MIT tarafından geliştirilmesi sürdürülmektedir. Kerberos v5 protokolünün doğrulanması için 3 farklı önerme kullanıldı. Bunlardan bir tane olası oturum önermesi olup kanıtlanmıştır. Bilet sunucusu ile istemcinin ortak şifresinin gizliliği ve kanıtlama önermeleri doğrulanamamıştır. Bu Tamarin'de kurulan modeldeki bir hatadan dolayı ya da protokolün kendisindeki bir hatadan dolayı olabilir. Bu sebeple Kerberos v5 protokolünün güvenliği ile ilgili bir yargıda bulunulamaz.

Tek kullanımlık şifre şeması, önceki oturumlarda kullanılan şifrelerden yenilerinin üretilmesi mantığı ile kanıtlama sağlayan bir protokoldür. Kayıt aşamasında ilk defa tek kullanımlık şifre oluşturulur ve güvenli bir kanal aracılığıyla sunucuya iletilir. Oturumun başarılı bir saldırıya uğraması için kayıt aşamasındaki verilerin gizliliğinin korunması gerekmektedir ancak bunlar güvenli kanal aracılığıyla gönderildiği için gizlilikleri varsayılmıştır. Bu sebeple sadece olası kayıt ve olası kanıtlama oturumlarının oluşması önermeleri dizayn edilmiştir. Bu önermelerin ikisi de kanıtlanmış olup protokolün güvenli olduğu sonucuna ulaşılmıştır.

Table of Contents

1 Introduction

Internet is used to build a channel between different end-users called as hosts. This channel does not offer security features by default. Packets that contain messages of each party in a communication through internet can easily be sniffed and had its contents fell into hands of adversaries without consent of any party. For this reason cryptographic protocols that satisfy different security properties such as secrecy, authentication, identification, authorization, confidentiality and integrity are implemented. These protocols are chosen for a specific scenario and formed according to the needs of security properties for that scenario and they denote a set of messages between (usually two) hosts and can be thought of steps, all parties have to satisfy. Parts of these messages consist of mathematical functions (hashing, encryption) and protocols grant security through them. But this security could be compromised even though the protocol got positive results from security analysis. This is only possible when the mathematical function is neither computationally secure nor unconditionally secure. There may be some cases where it was computationally secure at first but this may have changed by technological improvements (such as development of quantum computers making cryptanalysis of nowadays ciphers feasible that are infeasible for traditional computers) or later on, some shortcuts may be found in the mathematical functions making the computation trivial. Recent study on SHA1 shows that it is possible to generate practical collisions on SHA1 [1]. In this project, we are not interested in the security of mathematical functions. We will be interested in the verification of security protocols by analyzing its steps and messages. [2]

Verification of cryptographic protocols as its name suggests focuses on the protocols directly therefore mathematical functions are counted as unconditionally or computationally secure. Even if protocols can do the mentioned tasks, in some cases, it is possible that adversaries can work around the protocols and acquire or worse manipulate sensitive data. So, usage of unverified or flawed security protocols roughly same as not using any of them. The goal of the project is to find flaws in these protocols, fixing them before someone takes advantage of these deficiencies and results in monetary or private harm for uninformed individuals.

So, how is it possible for protocol to be verified? It could be done by humans as they can see situations that are hard to implement on computer easily, but they can never be sure whether they completed the verification or they thought they completed the verification. For this reason, verification process should be automated and computer tools should be used for having complete confidence whether the protocol is safe for work or not.

There are many different protocol verification tools to simulate and prove the protocols by writing appropriate mathematical lemmas. Some of these are AVISPA, Tamarin Prover, Isabelle[3], SecreC[4] and F*[5]. AVISPA and Tamarin Prover details will be explained below.

AVISPA (Automated Validation of Internet Security Protocol and Applications) is a automated theorem proving project that aims to provide a strong tool for this purpose, support and speed up the development of new internet protocols. AVISPA is the continuation of a project called The AVISS Project(Automated Validation of Infinite-State Systems) which started in 2001 lasting a year, it was a successful project because

it got attention to the automated theorem proving topic. AVISPA is a computation heavy security protocol modeling tool. As input to the program, along with the state machine of the protocol that is being verified, the variables that need to have security properties are specified along with which property they need [2].

Tamarin Prover is another security protocol verification tool which is published by Simon Meier and Benedikt Schmidt in 2012 as Doctorate thesis. But Tamarin Prover uses symbolic verification instead of a computation heavy verification. This is the part where Tamarin Prover differs from AVISPA tool. Symbolic verification prevents the tool from checking all possible situations. Not looking into all possible situations speeds up the process, because symbolic verification guides the tool to check more likely ones to be a solution. But this property makes the implementation complicated. In AVISPA, it is enough to state which variable needs which security property but, in Tamarin Prover, different lemmas have to be written for same security properties in different protocols.

2 Project Description and Plan

The scope of the project contains the verification of various protocols along with general working mechanisms of the protocols. Security protocols are complex topics that are implemented using various security concepts so an extensive literature research will be the first steps of the project plan. After having adequate knowledge of protocols, prover programs should be learned and protocols should be simulated on them. Following the completion of simulations, security lemmas should be implemented in security protocol models to check whether protocols are secure for work or not.

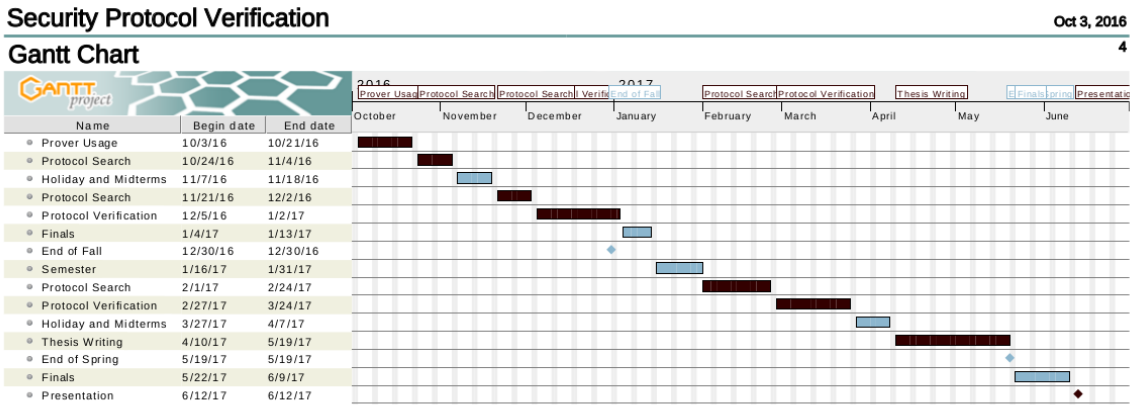

Figure 2.1: The Gantt diagram of the project

Since the project is about computer security, first thing to do is learn basic concepts of this topic. Computer security consists of basic cryptography, symmetric and asymmetric key algorithms, digital signatures and email, web and communication security. Inside Computer Networks book[6], this topics are clearly explained, so the book was the first address of initial research. After that, basic usage of Tamarin Prover, protocol modeling tool, will be learned. Without scanning different protocol, learning the tool does not make any difference, so various protocols will also be researched and tested on Tamarin Prover.

Although project process model looks like Waterfall approach, it is actually an iterative model which consists of doing more literature search and tests after gaining decent knowledge of the tools and concepts.

3 Background

Basic cryptography terminology, security properties, hash functions and one time pads are explained in the subsections below.

3.1 Public Key Cryptography

In public key cryptography, there are two keys: public and private keys. Everyone has access to public key of an individual but only that individual knows their private key. When Alice has a message for Bob, she encrypts the message with Bob's public key so that only Bob can read it by decrypting the ciphertext with his private key. For different implementations opposite scenario is also possible e.g. Alice can encrypt the message with her private key and when received Bob will use Alice's public key to decrypt ciphertext and read the message. This way everyone can read the message but Bob can be sure that Alice sent the message to himself since Alice signed it with her private key(non-repudiation).

3.2 Symmetric Key Cryptography

In symmetric key cryptography, there is only one key to encrypt and decrypt the data. The key is secret to those that have access to the system or message. The key must be pre-shared by all parties before or it must be exchanged online by methods like Diffie-Hellman Key Exchange. Since public key algorithms take more time compared to symmetric key algorithms due to immense calculations, data transmission is mostly done by symmetric keys after they are generated and exchanged by public key algorithms at the start of a session.

3.3 Security Properties

Security properties define different concepts and terms that security protocols may include.

3.3.1 CIA Triad

CIA Triad term was coined to guide organizations about which topics to take security measures. The elements of CIA are confidentiality, integrity and availability. These are the essentials for the security of computer systems and must be satisfied for all times[7].

3.3.1.1 Confidentiality

Confidentiality is the most understandable term among the CIA triad. It is about restricting the access of the data or system in question to those who are authorized[7]. To provide confidentiality, nowadays systems use encryption to isolate information to the authorized people by the use of passwords. Along with encryption, access permissions are also used to ensure confidentiality[8].

3.3.1.2 Integrity

Integrity is the term that corresponds to data being uncorrupted during its life cycle. The data must not change in storage or in transit without the consent of authorized parties[9]. Data corruption can be done by adversaries through different attack types but it is also plausible that natural hazards could happen like electric blackout or overheating in data centers (it is also possible that adversaries could do these by penetrating the systems)[7]. Precautions must be taken for these situations.

For the parts that are unrelated to hardware, it might not be possible to keep data uncorrupted but at least it is possible to detect these corruptions through hashing. If a host sends a hash of the message and shared password along with the message itself, adversary cannot generate a valid hash unless he knows the shared password. With a system like this, data in transit could be protected.

3.3.1.3 Availability

A system must be available when an authorized party needs to access the system. Given that nowadays systems have various services that many people use, this corresponds to being available at all times. Availability term also covers the reliability while accessing the system such as time needed to access the systems. If this value increases, system is counted unavailable. This may be caused by Distributed Denial of Service(DDoS) attacks that are done to cause economic damage to the companies by making their system inaccessible to potential customers. The system might be completely unavailable due to the power outages and natural hazards. Precautions must be taken to minimize these situations if not possible to prevent them[8].

3.3.2 Identification

Identification is a way to present one's identity to the system by supplying information like e-mail, nickname and social security number[10]. A person claims to be a specific person by identifying themself to the system. However identification does not cover one's proof of identity[11].

3.3.3 Authentication

Authentication is the process of verifying the claimed identity of a person received in the identification [10]. First, claimed identity is searched in the access list of the system, if there exists such identity, server sends a challenge to the host. This challenge could be about three different factors of authentication: Something the person knows, something the person has and something the person is. If the host responds the challenge with valid and right answer, host is authenticated[12].

In the first one, there could be many different challenges. The most popular one is the password for knowing factor. An answer to a prechosen security question, social security number, birthday and birthplace could be counted as alternatives.

The second factor is about persons belongings. Most popular example of authentication is done through messages sent to the persons mobile phone and requesting the

information in that message. Other frequently used items are ID cards, driving licenses and university student cards.

The last factor is about the data that defines that person. Fingerprint, retinal pattern and DNA are considered under this last factor.

3.3.4 Authorization

Authorization is the stage which occurs right after authentication phase ends[10]. There may be cases in which authorization is not implemented and once hosts are authenticated, they can access all of the system. But usually this is not the case, most of the time either authorization and authentication coexist together or neither of them exists[13]. Examples of the latter could be seen for most if not all websites. Each website has parts, all hosts can see. If authorization is implemented, hosts cannot access all of the services after authentication. Each and every host have access to different services and features, necessary checks are done while access request is sent by host. If hosts are in the access lists, they are authorized.

3.4 Hash Function

Hash function is a mathematical function that takes arbitrary size of input and returns fixed size of output. One way of using hash functions are using them to access data by indexing. This process decreases the time needed to access data and makes it possible for any type of data to be used as a key for accessing a value. They are called hash tables and used to implement key-value store data structures in programming languages.

Cryptographic hash functions are also made up from the same theory but they are made to satisfy different needs. In security, hash functions are used to answer the needs of message integrity. When a message is sent with its hash, one can be sure whether that message is modified or not by taking the hash of received message and comparing it with the received hash as long as the hash function used satisfies the factors given below. There are few factors that a function must satisfy to be called a good cryptographic hash function. These are as follows[14]:

- Given a hash value z = h(m), it must be computationally infeasible to find m
- Given z and m corresponding to z = h(m), it must be computationally infeasible to find m' which results in z = h(m')
- It must be computationally infeasible to find different messages that result in same hash values
- Taking hash of a value must be fast, but also must not be too fast because making a hash function fast also speeds up the bruteforce cryptanalysis of it.

3.5 One Time Pad

There exists a cryptographic algorithm that does not have any security problem in its theory and its called one time pad. If message to be sent XORed with a randomly

generated key that is at least long as the message and the result sent through the channel, a cryptanalyst cannot decipher the message even if he had all the computing power in the world. The reason behind it is that there is no information in the encrypted message because it has the maximum entropy. Even if he got the correct message, he can never be sure that is the correct message due to the fact that, one can choose a message and with another key, he can get to the same XOR result and that could be the real message. The workflow of the one time pads can be seen in 3.1

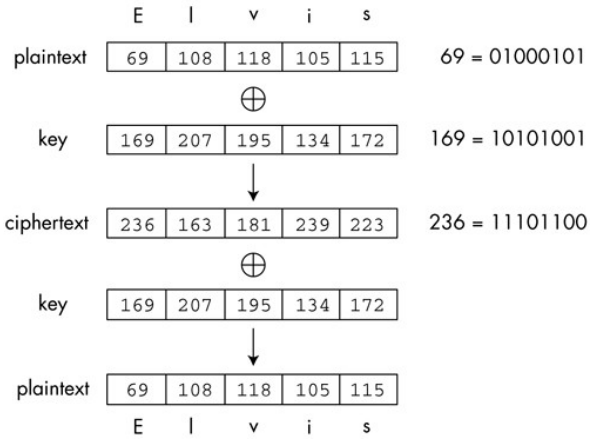

Figure 3.1: Workflow of one time pads [15]

The key being at least as long as the message makes one time pads impractical. Since both parties need to carry identical one time pads incase one of them needs to send a message, they must exchange the keys offline, otherwise one time pad will be as strong as the channel used to exchange the keys. If the cryptographic algorithm used on that channel is compromised, adversaries can easily access one time pads and decrypt the previous communications. To overcome this deficiency, one time password schemes are suggested. The idea behind one time password schemes are about calculating one time pad for the next session based on previous ones. If both parties exchange keys once through a secure channel, they can use one time pads forever. There are different ways to make a one time password scheme but in the paper[16], a mathematical function is used to generate next one time pads. And that scheme is explained in the implementation section.

4 Analysis and Modeling

In this section tools used and protocols modeled are analyzed and explained in detail.

4.1 Tools

Only the the Tamarin Prover tool is used but modeled protocols are adapted from AVISPA models and AVISPA tool is also introduced.

4.1.1 Tamarin Prover

Tamarin Prover is a tool that is used to simulate security protocols and analyze them by different lemmas to deduce whether a certain security property holds or not. This is done by taking security protocol models as inputs and defining the actions that different parties (adversary, client or server) can do[17]. There are different tools that are used for security protocol modeling and verification (AVISPA is one of them along with Tamarin Prover.). However Tamarin Prover has a great aspect which is visual support that is the main reason it is preferred for this project. The protocol, its rules, axioms and lemmas are all visualized on browser. The proof of protocol model can be done step by step on this visual interface which simplifies understanding and speed up the process.

4.1.2 AVISPA

AVISPA is a project that is started to speed up the development of new internet protocols by automating their security verification and finding the flaws and fixing them iteratively[2]. Finding security flaws by looking or on the fly is a time consuming process because there must be too many tests to cover all cases. By simulating the protocol with its state machine on AVISPA, it takes much less time since it checks all possible situations by itself. Since AVISPA is a computational tool, one can be sure of its security if AVISPA states that protocol is secure after all possible situations are simulated. But in some cases, it may not be possible to try all possible situations if it takes forever to compute them. In these cases tools like Tamarin Prover would be the address for automated protocol verification.

4.2 Protocols

CRAM-MD5, APOP, CHAPv2, Kerberos v5 and Isawa and Morii's OTP Scheme[16] are modeled, so foundations of these protocols are described in the subsections below.

4.2.1 CRAM-MD5

CRAM-MD5 is a Challenge-Response Authentication Mechanism which presents an authentication extension to IMAP4 that does not transfer keys in plaintext and does not require advanced security infrastructure to work [18]. CRAM-MD5 was designed to prevent plaintext password transmission. Given that an adversary Eve can listen to the conversation, all Eve will be able to get is the challenge and hashed response with the password [19]. Hashing algorithms usually work fast but it is bad for hashing algorithms to be too fast, since being fast speeds up the brute-force cryptanalysis,

so a slow function must be used. CRAM-MD5 checks for authentication only in the authentication step, so adversary that can impersonate a person can replace the person after the authentication step and apply any command they desire.

CRAM-MD5 is simulated as a security protocol model on Tamarin Prover tool and proved with appropriate lemmas.

4.2.2 APOP

POP (Post Office Protocol) is an old mail service protocol that is used to retrieve mails from the server to e-mail client and deletes them from server after client retrieves them[20]. In IMAP(Internet Message Access Protocol) e-mails are kept in remote server even after they are accessed from an e-mail client. This allows hosts to use different e-mail client applications and still see same e-mails. IMAP is more suitable to today's needs given that most of the population have more than one way (like computers and smartphones) to check e-mails.

POP authenticates a client with its username and password but these data are sent over internet in plaintext [21]. This gives access of all future e-mails to anyone who listens to the POP authentication session. In APOP (Authenticated Post Office Protocol), client sends authentication request with username included. After handshake is completed, client encrypts the password and sends encrypted password through the channel. On receival, server decrypts the password and validates the client.

4.2.3 CHAPv2

CHAP (Challenge-Handshake Authentication Protocol) is an authentication protocol that is used to certify identity of remote clients. CHAP is preceded by PAP(Password Authentication Protocol) which is another authentication protocol but it sends unencrypted passwords through the channel and due to this reason, it is considered a weak authentication scheme. PPP(Point-to-Point Protocol) uses CHAP (if CHAP is not supported by both parties, it uses PAP) in its authentication phase and the process is repeated at random intervals with random challenge strings to prevent replay attack. MS-CHAPv1 and MS-CHAPv2 are both modified versions of CHAP according to the needs of Microsoft to make authenticated sessions between client and Microsoft servers. In CHAPv1, one way authentication is present with complete backwards compatibility to CHAP. In CHAPv2, mutual authentication is present in which both parties certify the other party asking a challenge and matching response with the calculated result of the challenge[22].

4.2.4 Kerberos v5

Kerberos is a protocol that is designed to satisfy network authentication needs of campus connection of MIT (Massachusetts Institute of Technology). Therefore, it is designed and developed by MIT as a part of Project Athena. First 3 versions (Kerberos v1, v2 and v3) are only used in MIT network. Kerberos v4 was developed in late 1980s. Version 4 used DES as its main algorithm for security purposes. Since DES uses 56-bit keys, it is no longer secure to use DES in cryptographic operations due to the advances in the computer performances. Apart from the fact that Version 4 uses DES, there are

some security flaws in Version 4 that allows cryptanalysts to find keys without having to do brute-force search on complete key set [23]. For these reasons, MIT dropped support for Version 4 and recommends servers to migrate to Kerberos v5 that was developed in 1993 to patch security flaws of Kerberos v4. Kerberos v5 supports many different cryptographic algorithms and still being supported by MIT Kerberos Team.

4.2.5 Isawa and Morii's OTP Scheme

One time pads are unconditionally secure unless they are shorter than the message being sent. This feature of one time pads make them invaluable for cryptographic processes however there is a tradeoff. For example, if a host needs to send 5GBs of data, it also needs 5GBs of OTP along with it. And it cannot reuse the same OTP more than once (as name suggests) because that would ruin the unconditional security feature by decreasing entropy and making cryptanalysis possible.

One time password schemes are cryptographic protocols that utilize one time pads to provide security. There are different ways to make one time password schemes one of which is using a mathematical function that takes input a one time pad and generates another one time pad [16] and this method is used in the scheme explained in implementation section. This saves hosts from managing big chunks of one time pad for every message they will send and receive.

5 Design and Implementation

In this section model details of protocols are explained.

5.1 CRAM-MD5

In CRAM-MD5, first client sends its identity to server. Once server receives the identity of client, it generates timestamp and nonce and send them back to client with its identity included. Client then sends the hashed shared key to server with the timestamp to complete the authentication.

1: $\mathbf{A} \rightarrow \mathbf{B} : A$
2: $\mathbf{B} \rightarrow \mathbf{A} : Ns.T.S$
3: $\mathbf{A} \rightarrow \mathbf{B} : F(SK.T)$

Algorithm 5.1: Alice Bob Notation of CRAM-MD5 protocol [18]

Ns denotes nonce generated by the server, T is a timestamp, SK is the shared key between A and S, F is the MD-5 hash function.

5.2 APOP

In APOP, first server sends a hello message along with the timestamp. After client receives the message, he concatenates timestamp with the shared key K_CS and hashes the result. Then he concatenates his ID with the hash and sends it back to server. When server receives the message, server calculates hash and checks the validity of the hash sent by client. If they match, server sends "Success" message.

1: $\mathbf{S} \rightarrow \mathbf{C} : Hello.Timestamp$
2: $\mathbf{C} \rightarrow \mathbf{S} : C.MD5(Timestamp.K_CS)$
3: $\mathbf{S} \rightarrow \mathbf{C} : Success$

Algorithm 5.2: Alice Bob Notation of APOP [24]

MD5 denotes the hashing algorithm used in the protocol. K_CS is the shared key between client C and server S.

5.3 CHAPv2

In CHAPv2, Alice sends her identity to Bob and Bob responds with challenge "N_b". After receiving "N_b", Alice hashes shared key k(A, B), "N_a", "N_b" and her identity "A". Then she sends the hash with "N_a" challenge. Bob calculates the hash same way and matches them to check whether they are equal or not. If they are equal, Bob hashes shared key k(A, B) and challenge "N_a" and sends the hash to Alice. If there is a mismatch Bob terminates the session. When Alice receives the hash Bob calculated, she calculates the hash with same operations and checks if they are equal. If they are equal, mutual authentication is satisfied, otherwise session is aborted.

1: $\mathbf{A} \rightarrow \mathbf{B} : A$
2: $\mathbf{B} \rightarrow \mathbf{A} : N_b$
3: $\mathbf{A} \rightarrow \mathbf{B} : N_a, H(k(A,B),(N_a,N_b,A))$
4: $\mathbf{B} \rightarrow \mathbf{A} : H(k(A,B),N_a)$

Algorithm 5.3: Alice Bob Notation of MS-CHAPv2 [25]

5.4 Kerberos v5

First, client C sends a message to authentication server A containing its username, the server it wants to connect, requested lifetime and network address. After receiving the message, A checks if the username exists in the users list and terminates the protocol if it is not so. Then A generates two messages one of them is an encrypted ticket that only Ticket Granting Server G can decrypt. It contains C, G, a session key between C and G along with validity period of the ticket. Second message is encrypted with K_{CA}(known as clients secret key) and contains G, session key between C and G, validity period and network address. At this point C has the Ticket Granting Ticket and ready to talk with G.

Secondly, C sends two messages to G. First of them is an authentication message containing username of C and timestamp encrypted with session key between C and G. Contents of the second message are ID of the HTTP service server, requested lifetime, network address and the encrypted ticket from the session with the authentication server. Once G gets the messages, it decrypts the ticket and get the shared session key between C and G. With the session key it decrypts authentication message and matches the identities inside. Then it checks whether the message is expired or not. Also it checks if the authentication message is processed before to prevent replay attacks. After all of the checks, G too generates two messages. One of them is encrypted with K_{CG} and contains ID of the HTTP service server, session key K_{CS}, validity period of message and network address. This message is for the C to decrypt. Other message is encrypted with shared key between server G and service server S. It contains C, S, session key between C and S and validity period of the message.

Lastly, C makes a connection between HTTP service server S. C decrypts the message with K_{CG} and acquires new session key K_{CS}, session key between C and S, and forwards the ticket received during previous session to S. C sends authentication message to S, encrypted with the session key between them containing username of C and timestamp. Upon receiving the message, S decrypts the ticket with K_{GS} shared key and learns the session key K_{CS}. With K_{CS}, S decrypts the authentication message and matches the identities in them and checks whether messages are still valid. S also makes the replay checks to see if same authentication message is received earlier. After all the checks pass, S sends an encrypted message containing last received timestamp. C decrypts that message and check if sent and received timestamps match. If there is no problem in the above steps, mutual authentication is satisfied. Authentication is complete and client C now can use the services of HTTP server S before ticket times out [26].

1: $\mathbf{C} \rightarrow \mathbf{A} : C, G, Lifetime_1, N_1$
2: $\mathbf{A} \rightarrow \mathbf{C} : C, Ticket_1, (G, K_{CG}, Tstart, Texpire, N_1)K_{CA}$
3: $\mathbf{C} \rightarrow \mathbf{G} : S, Lifetime_2, N_2, Ticket_1, (C, T)K_{CG}$
4: $\mathbf{G} \rightarrow \mathbf{C} : C, Ticket_2, (S, K_{CS}, Tstart2, Texpire2, N_2)K_{CG}$
5: $\mathbf{C} \rightarrow \mathbf{S} : Ticket_2, (C, T_2)K_{CS}$
6: $\mathbf{S} \rightarrow \mathbf{C} : (T_2)K_{CS}$
7: $Macros :$
8: $Ticket_1 = (C, G, K_{CG}, Tstart, Texpire)K_{AG}$
9: $Ticket_2 = (C, S, K_{CS}, Tstart_2, Texpire_2)K_{GS}$

Algorithm 5.4: Alice Bob Notation of Kerberos v5 [27]

5.5 Isawa and Morii's OTP Scheme

Isawa and Morii introduced a one time password scheme[16] that depends on calculating next authentication keys from earlier session authentication keys. These keys are generated as one time pads which means they are unconditionally secure.

5.5.1 Registration Phase

All messages sent in registration phase use secure channel as communication medium. First user U chooses an ID and password PW and sends his ID. Then, server S produces three random numbers R_0, R_{-1}, F_0 and sends them. U calculates the values listed in the registration phase macros. As last message of this phase U sends F_1 and V_1 to S.

1: $\mathbf{U} \Rightarrow \mathbf{S} : ID$
2: $\mathbf{S} \Rightarrow \mathbf{U} : R_0, R_{-1}, F_0$
3: $\mathbf{U} \Rightarrow \mathbf{S} : F_1, V_1$
4: $RegistrationPhaseMacros :$
5: $A_1 = h(ID||PW||F_0)$
6: $F_1 = h(A_1)$
7: $Q_1 = h(PW||R_{-1})$
8: $A_2 = h(ID||Q_1||F_1)$
9: $F_2 = h(A_2)$
10: $Q_2 = h(Q_1||R_0)$
11: $V_1 = h(A_1||F_2)$
12: $UStored : ID, Q_2, A_1, F_1, A_2, F_2$
13: $SStored : ID, F_1, V_1$

Algorithm 5.5: Alice Bob Notation of Registration Phase of One Time Password Scheme [16]

5.5.2 Authentication Phase

All messages sent in authentication phase use insecure channel as communication medium. First U calculates the values denoted by authentication phase macros in the table below. Then, U sends F_{i+1} XOR F_i, A_i XOR F_{i+1}, V_{i+1} and h($F_i.V_{i+1}$) to S. For the validity of the F_{i+1} and A_i, S uses F_i that is stored in registration phase to do the following operations:

1: $\mathbf{U} \rightarrow \mathbf{S}: F_{i+1} \oplus F_i, A_i \oplus F_{i+1}, V_{i+1}, h(F_i||V_{i+1})$
2: $\mathbf{S} \rightarrow \mathbf{U}: R_i, h(R_i||F_i)$
3: $AuthenticationPhaseMacros:$
4: $A_{i+2} = h(ID||Q_{i+1}||F_{i+1})$
5: $F_{i+2} = h(A_{i+2})$
6: $F_{i+1} \oplus F_i$
7: $A_i \oplus F_{i+1}$
8: $V_{i+1} = h(A_{i+1}||F_{i+2})$
9: $h(F_i||V_{i+1})$
10: $UStored: Q_{i+2}, A_{i+2}, F_{i+2}$
11: $SStored: F_{i+1}, V_{i+1}$

Algorithm 5.6: Alice Bob Notation of Authentication Phase of One Time Password Scheme [16]

6 Testing and Evaluation

In this section, for each protocol modeled, lemmas are reported with the results of the executions included.

6.1 CRAM-MD5

Listing 6.1: Implementation of secrecy lemma of CRAM-MD5 protocol

```
lemma SK_secrecy:
  "
    not(
    Ex A S SK T #i #j.
      Session( A, S, SK, T ) @ #i
    & K(SK) @ #j
    & not( Ex #r. SK_Reveal(A, S) @ #r)
    )
  "
```

Listing 6.2: Execution results are positive for secrecy lemma of CRAM-MD5 protocol

```
lemma SK_secrecy:
  all-traces
  "not(Ex A S SK T #i #j.
            ((Session( A, S, SK, T ) @ #i) and (K( SK ) @ #j)) and
            (not(Ex #r. SK_Reveal( A, S ) @ #r)))"
simplify
solve( !Shared_Key( A, S, SK ) #i )
  case Register_SK
  solve( Server_1( T )  #i )
    case Server_1
    solve( !KU( ~SK ) @ #vk.1 )
      case Reveal_SK
      by contradiction /* from formulas */
    qed
  qed
qed
```

Once A receives the challenge message containing timestamp, they send back the hash of timestamp and shared key. So if an adversary knows the shared key, they can authenticate themself by using it. This is why secrecy of shared key must be ensured. Result computed with the lemma given above shows that secrecy of shared key is ensured.

Listing 6.3: Implementation of authentication lemma of CRAM-MD5 protocol

```
lemma Client_Authentication:
  "
  (  All A S SK T #i. Session( A, S, SK, T ) @ #i
    ==>
    (
      (( Ex #a. AnsweredRequest(S,SK) @ a )
         & (All #j. Session(A,S,SK,T) @ #j ==> #i = #j) )
      | ( Ex #r. SK_Reveal(A, S) @ r & r < i )
    )
  )
  "
```

If a session is created successfully, it is so that either shared key is revealed or the host being authenticated is the one that they claimed to be and that host answered the challenge requested by server.

Listing 6.4: Execution results are positive for authentication lemma of CRAM-MD5 protocol

```
lemma Client_Authentication:
  all-traces
  "All A S SK T #i.
         (Session( A, S, SK, T ) @ #i) ==>
         (((Ex #a. AnsweredRequest( S, SK ) @ #a) and
           (All #j. (Session( A, S, SK, T ) @ #j) ==> (#i = #j))) or
          (Ex #r. (SK_Reveal( A, S ) @ #r) and (#r < #i)))"
simplify
solve( (All #a. (AnsweredRequest( S, SK ) @ #a) ==> cont) ||
       (Ex #j. (Session( A, S, SK, T ) @ #j) and not(#i = #j)) )
  case case_1
  by contradiction /* from formulas */
next
  case case_2
  solve( (#i < #j) or (#j < #i) )
    case case_1
    solve( !Shared_Key( A, S, SK ) #i )
      case Register_SK
      solve( Server_1( T ) #i )
        case Server_1
        solve( !Shared_Key( $A, $S, ~SK ) #j )
          case Register_SK
          solve( Server_1( ~T ) #j )
            case Server_1
            by contradiction /* cyclic */
          qed
        qed
      qed
    qed
  next
    case case_2
    solve( !Shared_Key( A, S, SK ) #i )
      case Register_SK
      solve( Server_1( T ) #i )
        case Server_1
        solve( !Shared_Key( $A, $S, ~SK ) #j )
          case Register_SK
          solve( Server_1( ~T ) #j )
            case Server_1
            by contradiction /* cyclic */
          qed
```

Listing 6.5: Implementation of session key possible setup lemma of CRAM-MD5 protocol

```
lemma session_key_setup_possible:
  exists-trace
  " /* There is a trace satisfying all equality checks */
   (All x y #i. Eq(x,y) @ i ==> x = y)
  &  /* Session keys have been setup */
   (Ex A S SK T #k.  Session(A, S, SK, T) @ k
    /* without having performed a long-term key reveal. */
    & not (Ex #r. SK_Reveal(A, S) @ r)
    )
  "
```

Possible setup lemmas exists just for sanity check. Given the current protocol rules, they make the check whether there is a possible situation to complete a session with all equality checks succeed or not. Normally, all lemmas are executed with the (all-traces) condition which states that given lemma must be true under all situations. But, in possible setup lemmas, (exists-trace) condition is used to ensure that there is at least one situation in which protocol can successfully terminate.

Listing 6.6: Execution results are positive for session key possible setup lemma of CRAM-MD5 protocol

```
lemma session_key_setup_possible:
  exists-trace
  "(All x y #i. (Eq( x, y ) @ #i) ==> (x = y)) and
        (Ex A S SK T #k.
          (Session( A, S, SK, T ) @ #k) and
          (not(Ex #r. SK_Reveal( A, S ) @ #r)))"
simplify
solve( !Shared_Key( A, S, SK ) #k )
  case Register_SK
  solve( Server_1( T ) #k )
    case Server_1
    solve( !KU( h(<~SK, ~T>) ) @ #vk )
      case Client_2
      solve( !KU( ~T ) @ #vk.5 )
        case Server_1
        SOLVED // trace found
      qed
    qed
  qed
qed
```

6.2 APOP

Listing 6.7: Implementation of shared key secrecy lemma of APOP

```
lemma SK_secrecy:
  " /* It cannot be that a */
  not(
    Ex S C SK T #i #j.
      Successful( S, C, SK, T ) @ #i
    & K(SK) @ #j
    & not( Ex #r. SK_Reveal(S, C) @r)
  )
  "
```

Listing 6.8: Execution results are positive for shared key secrecy lemma of APOP

```
lemma SK_secrecy:
  all-traces
  "not(Ex S C SK T #i #j.
          ((Successful( S, C, SK, T ) @ #i) and (K( SK ) @ #j)) and
          (not(Ex #r. SK_Reveal( S, C ) @ #r)))"
simplify
solve( !Shared_Key( S, C, SK ) #i )
  case Register_SK
  solve( Server_1( T ) #i )
    case Server_1
    solve( !KU( ~SK ) @ #vk.1 )
      case Reveal_SK
      by contradiction /* from formulas */
    qed
  qed
qed
```

Since timestamp is authenticated with the shared key between client and server, its secrecy must be ensured. The result in 6.8 shows that, secrecy of the shared key is ensured.

Listing 6.9: Implementation of strong authentication lemma of APOP

```
lemma Authentication:
  "
    ( All S C SK T #i. AuthDone( S, C, SK, T ) @ #i
      ==>
       ( (Ex #r. SK_Reveal(S, C) @ r & r < i)
        | (Ex #l. AnswerReq(S, C, SK, T) @l
        & (All #j. AuthDone( S, C, SK, T ) @ #j ==>  #i = #j )
        )
```

```
        )
    )
  "
```

The goal of APOP is to provide authentication and replay protection, so instead of weak authentication, strong authentication lemma is used. The different between the two is that there is an extra check in the strong authentication lemma. This check guarantees that an authentication is not replayed by matching all of its arguments with another authentication, if they match and they are not identical, protocol detects a replay attack.

Listing 6.10: Execution results are positive for strong authentication lemma of APOP

```
lemma Authentication:
  all-traces
  "All S C SK T #i.
         (AuthDone( S, C, SK, T ) @ #i) ==>
         ((Ex #r. (SK_Reveal( S, C ) @ #r) and (#r < #i)) or
          (Ex #l.
            (AnswerReq( S, C, SK, T ) @ #l) and
            (All #j. (AuthDone( S, C, SK, T ) @ #j) ==> (#i = #j))))"
simplify
solve( !Shared_Key( S, C, SK ) #i )
  case Register_SK
  solve( Server_1( T ) #i )
    case Server_1
    solve( !KU( h(<~T, ~SK>) ) @ #vk.2 )
      case Client_1
      solve( (#i < #j)|| (#j < #i) )
        case case_1
        solve( !Shared_Key( $S, $C, ~SK ) #j )
          case Register_SK
          solve( Server_1( ~T ) #j )
            case Server_1
            by contradiction /* cyclic */
          qed
        qed
      next
        case case_2
        solve( !Shared_Key( $S, $C, ~SK ) #j )
          case Register_SK
          solve( Server_1( ~T ) #j )
            case Server_1
            by contradiction /* cyclic */
          qed
        qed
      qed
    next
      case ch
      solve( !KU( ~SK ) @ #vk.5 )
        case Reveal_SK
        by contradiction /* from formulas */
      qed
    qed
  qed
qed
```

Listing 6.11: Implementation of possible setup lemma of APOP

```
lemma session_key_setup_possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @ i ==> x = y)
  &
  (Ex S C SK T #j. Successful(S, C, SK, T) @ #j
    & not( Ex #r. SK_Reveal(S, C) @r )
  )
  "
```

As explained in the previous section, this lemma exists to prove that protocol works under at least one situation.

Listing 6.12: Execution results are positive for possible setup lemma of APOP

```
lemma session_key_setup_possible:
  exists-trace
  "(All x y #i. (Eq( x, y ) @ #i) ==> (x = y)) and
        (Ex S C SK T #j.
          (Successful( S, C, SK, T ) @ #j) and
          (not(Ex #r. SK_Reveal( S, C ) @ #r)))"
simplify
solve( !Shared_Key( S, C, SK ) #j )
  case Register_SK
  solve( Server_1( T ) #j )
    case Server_1
    SOLVED // trace found
  qed
qed
```

6.3 CHAPv2

Listing 6.13: Implementation of shared key secrecy lemma of CHAPv2 protocol

```
lemma SK_secrecy:
  " /* It cannot be that a */
  not(
    Ex Na Nb A B SK_AB #i #j.
      Session( Na, Nb, A, B, SK_AB ) @ #i
    & K(SK_AB) @ #j
    & not( Ex #r. SK_Reveal(A,B) @r)
  )
  "
```

Listing 6.14: Execution results are positive for shared key secrecy lemma of CHAPv2 protocol

```
lemma SK_secrecy:
  all-traces
  "not(Ex Na Nb A B SK_AB #i #j.
            ((Session( Na, Nb, A, B, SK_AB ) @ #i) and (K( SK_AB ) @ #j)) and
            (not(Ex #r. SK_Reveal( A, B ) @ #r)))"
simplify
solve( Client_2( Na, Nb, A, B, SK_AB ) #i )
  case Client_2
  solve( !KU( ~SK_AB ) @ #vk.1 )
    case Reveal_SK
    by contradiction /* from formulas */
  qed
qed
```

In this protocol, client and server share a secret key and using that key they send each other challenges and respond to them with hashes of secret key and the challenge. Since both of them tries to authenticate each other, mutual authentication is done. Due to these steps, secrecy of shared key must be ensured and according to the result it is ensured.

Listing 6.15: Implementation of N_a strong authentication lemma of CHAPv2 protocol

```
lemma Auth_on_Na:
  "
  ( All A B SK_AB Na #i. Auth_Na(A, B, SK_AB, Na) @ #i
    ==>
    ( (Ex #j. SK_Reveal(A, B) @ j & j < i)
      | ( Ex #r. Server_Answered(A, B, Na, SK_AB) @r
```

```
        &  (All #k. Auth_Na(A, B, SK_AB, Na) @ #k ==> #i = #k   )
        )
    )
  )
  "
```

Listing 6.16: Execution results are positive for N_a strong authentication lemma of CHAPv2 protocol

```
lemma Auth_on_Na:
  all-traces
  "All A B SK_AB Na #i.
         (Auth_Na( A, B, SK_AB, Na ) @ #i) ==>
         ((Ex #j. (SK_Reveal( A, B ) @ #j) and (#j < #i)) or
          (Ex #r.
            (Server_Answered( A, B, Na, SK_AB ) @ #r) and
            (All #k. (Auth_Na( A, B, SK_AB, Na ) @ #k) ==> (#i = #k))))"
simplify
solve( Client_2( Na, Nb, A, B, SK_AB ) #i )
  case Client_2
  solve( !KU( h(<~Na, ~SK_AB>) ) @ #vk )
    case Server_2
    solve( (#i < #k) || (#k < #i) )
      case case_1
      solve( Client_2( ~Na, Nb.1, $A, $B, ~SK_AB ) #k )
        case Client_2
        by contradiction /* cyclic */
      qed
    next
      case case_2
      solve( Client_2( ~Na, Nb.1, $A, $B, ~SK_AB ) #k )
        case Client_2
        by contradiction /* cyclic */
      qed
    qed
  next
    case ch
    solve( !KU( ~SK_AB ) @ #vk.4 )
      case Reveal_SK
      by contradiction /* from formulas */
    qed
  qed
qed
```

Authentication of the nonce N_a is checked with strong authentication lemma and according to the result, authentication is established.

Listing 6.17: Implementation of N_b strong authentication lemma of CHAPv2 protocol

```
lemma Auth_on_Nb:
  "
  ( All A B SK_AB Nb #i. Auth_Nb  (A, B, SK_AB, Nb) @ #i
    ==>
    (
      (Ex #j. SK_Reveal(A, B) @ j & j < i)
      | (Ex #r. Client_Answered(A, B, Nb, SK_AB) @r
      & (All #k. Auth_Nb  (A, B, SK_AB, Nb) @ #k ==> #i = #k))
    )
  )
  "
```

Listing 6.18: Execution results are positive for N_b strong authentication lemma of CHAPv2 protocol

```
lemma Auth_on_Nb:
  all-traces
  "All A B SK_AB Nb #i.
         (Auth_Nb( A, B, SK_AB, Nb ) @ #i) ==>
         ((Ex #j. (SK_Reveal( A, B ) @ #j) and (#j < #i)) or
```

```
          (EX #r.
            (Client_Answered( A, B, Nb, SK_AB ) @ #r) and
            (All #k. (Auth_Nb( A, B, SK_AB, Nb ) @ #k) ==> (#i = #k))))"
simplify
solve( !Shared_Key( A, B, SK_AB ) #i )
  case Register_SK
  solve( Server_1( Nb ) #i )
    case Server_1
    solve( !KU( h(<~SK_AB, z, ~Nb, $A>) ) @ #vk.2 )
      case Client_2
      solve( (#i < #k) || (#k < #i) )
        case case_1
        solve( !Shared_Key( $A, $B, ~SK_AB ) #k )
          case Register_SK
          solve( Server_1( ~Nb ) #k )
            case Server_1
            by contradiction /* cyclic */
          qed
        qed
      next
        case case_2
        solve( !Shared_Key( $A, $B, ~SK_AB ) #k )
          case Register_SK
          solve( Server_1( ~Nb ) #k )
            case Server_1
            by contradiction /* cyclic */
          qed
        qed
      qed
    next
      case ch
      solve( !KU( ~SK_AB ) @ #vk.6 )
        case Reveal_SK
        by contradiction /* from formulas */
      qed
    qed
  qed
qed
```

Authentication of the nonce N_b is checked with strong authentication lemma and according to the result, authentication is established.

Listing 6.19: Implementation of possible setup lemma of CHAPv2 protocol

```
lemma session_key_setup_possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @ i ==> x = y)
  &
  ( Ex Na Nb A B SK_AB #r. Session( Na, Nb, A, B, SK_AB ) @ #r
    & not( Ex #j. SK_Reveal(A, B) @j )
  )
  "
```

Listing 6.20: Execution results are positive for possible setup lemma of CHAPv2 protocol

```
lemma session_key_setup_possible:
  exists-trace
  "(All x y #i. (Eq( x, y ) @ #i) ==> (x = y)) and
        (Ex Na Nb A B SK_AB #r.
          (Session( Na, Nb, A, B, SK_AB ) @ #r) and
          (not(Ex #j. SK_Reveal( A, B ) @ #j)))"
simplify
solve( Client_2( Na, Nb, A, B, SK_AB ) #r )
  case Client_2
  solve( !KU( h(<~Na, ~SK_AB>) ) @ #vk )
    case Server_2
    solve( !KU( h(<~SK_AB, ~Na, ~Nb.1, $A>) ) @ #vk.4 )
      case Client_2
      solve( !KU( ~Na ) @ #vk.4 )
```

```
        case Client_2
        solve( !KU( ~Nb ) @ #vk.4 )
          case Server_1
          SOLVED // trace found
        qed
      qed
    qed
  qed
qed
```

Possible setup lemma is implemented as given in 6.19 and it proves that protocol terminates successfully under at least one situation.

6.4 Kerberos v5

Listing 6.21: Implementation of possible setup lemma of Kerberos v5 protocol

```
lemma Session_Key_Setup_Possible:
  exists-trace
  "
  Ex C S params #j. Session(C, S, params) @j
   & not( Ex A #r. SK_Reveal(C,A) @r )
   & not( Ex A G #s. SK_Reveal(A,G) @s )
   & not( Ex G #t. SK_Reveal(G,S) @t )
  "
```

Listing 6.22: Execution results are positive for possible setup lemma of Kerberos v5 protocol

```
lemma Session_Key_Setup_Possible:
  exists-trace
  "Ex C S params #j.
         (((Session( C, S, params ) @ #j) and
           (not(Ex A #r. SK_Reveal( C, A ) @ #r))) and
          (not(Ex A G #s. SK_Reveal( A, G ) @ #s))) and
         (not(Ex G #t. SK_Reveal( G, S ) @ #t))"
simplify
solve( !Parties( A, C, G, S ) #j )
  case Parties
  solve( !KU( senc(T2, Key_CS) ) @ #vk )
    case AuthServ_1_case_1
    SOLVED // trace found
  qed
qed
```

Possible setup lemma proved that protocol can terminate successfully in at least one situation.

Listing 6.23: Implementation of session key CG secrecy lemma of Kerberos v5 protocol

```
lemma CG_secrecy:
  " /* It cannot be that a */
  not(
    Ex C G A k #i #j.
      /* client has set up a session key 'k' with a server 'S' */
      SessKeyCG(C, G, A, k) @ #i
      /* and the adversary knows 'k' */
    & K(k) @ #j
      /* without having performed a long-term key reveal on 'S'. */
    & not( (Ex #r. SK_Reveal(A, G) @ r) | (Ex #l. SK_Reveal(C, A) @ l) )
  )
  "
```

Listing 6.24: Execution results are negative for session key CG secrecy lemma of Kerberos v5 protocol

```
lemma CG_secrecy:
  all-traces
  "not(Ex C G A k #i #j.
          ((SessKeyCG( C, G, A, k ) @ #i) and (K( k ) @ #j)) and
          (not((Ex #r. SK_Reveal( A, G ) @ #r) and
             (Ex #l. SK_Reveal( C, A ) @ #l))))"
simplify
solve( !Parties( A, C, G, S ) #i )
  case Parties
  solve( !Shared( $G, $S, Key_GS ) #i )
    case Register_SK
    solve( !KU( senc(<$C, ~T>, k) ) @ #vk.8 )
      case Client_2
      solve( !KU( senc(<$G.1, k, Tstart, Texpire, N_1>, Key_CA)
             ) @ #vk.14 )
        case AuthServ_1_case_1
        SOLVED // trace found
      qed
    qed
  qed
qed
```

Secrecy of session key Key_CG is important because once an adversary learns the Key_CG, they can impersonate the client for the rest of the protocol and connect the server on behalf of the client. As it can be seen from the figure 6.24, secrecy of Key_CG is not ensured. This may be due to the poor implementation of the Kerberos protocol or there may actually be a security flaw in the Kerberos v5 protocol.

Listing 6.25: Implementation of weak authentication lemma of Kerberos v5 protocol

```
lemma CG_weak_auth:
  " /* For all session keys 'k' setup by clients with a server 'S' */
    ( All C G A k #i. SessKeyCG(C, G, A, k) @ #i
    ==>
      /* there is a server that answered the request */
      ( (Ex #a. Answer_CG(C, G, A, k) @ a)
      /* or the adversary performed a long-term key reveal on 'S'
      before the key was setup. */
      | (Ex #r. SK_Reveal(A, G) @ r & r < i)
      | (Ex #l. SK_Reveal(C, A) @ l & l < i)
      )
    )
  "
```

Listing 6.26: Execution results are negative for weak authentication lemma of Kerberos v5 protocol

```
lemma CG_weak_auth:
  all-traces
  "All C G A k #i.
         (SessKeyCG( C, G, A, k ) @ #i) ==>
         (((Ex #a. Answer_CG( C, G, A, k ) @ #a) or
           (Ex #r. (SK_Reveal( A, G ) @ #r) and (#r < #i))) or
          (Ex #l. (SK_Reveal( C, A ) @ #l) and (#l < #i)))"
simplify
solve( !Parties( A, C, G, S ) #i )
  case Parties
  solve( !Shared( $G, $S, Key_GS ) #i )
    case Register_SK
    solve( !KU( senc(<$C, ~T>, k) ) @ #vk.8 )
      case Client_2
      solve( !KU( senc(<$G.1, k, Tstart, Texpire, N_1>, Key_CA)
             ) @ #vk.13 )
        case AuthServ_1_case_1
        SOLVED // trace found
      qed
    qed
  qed
qed
```

Authentication of Key_CG is not ensured as it is denoted in the figure caption of 6.26. Same as the previous lemma, this also may be due to the poor implementation of the protocol or there may actually be flaws in the Kerberos v5 protocol.

6.5 One Time Password Scheme

One time password scheme is a complex protocol due to many steps that exists to prevent various attack types. Since registration phase is done through a secure channel, there is no need to verify it. Authentication phase depends on the registration phase variables ID, PW, R_0, R_{-1} and F_0. Secrecy of these variables are already ensured due to the usage of secure channel. Even if these variables are compromised, one should know all of the random R_i values that are transmitted during the authentication phases to make a successful attack. Due to these reasons only the possible setup lemmas are implemented.

Listing 6.27: Implementation of authentication possible lemma of One Time Password Scheme

```
lemma Authentication_Possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @i ==> x = y)
  &
  ( Ex U S V1 A1 F2 #j. !S_Authenticates_U(S,U,V1,A1,F2) @j
    & Ex R1 F1 hRF #k. !U_Authenticates_S(U,S,R1,F1,hRF) @k )
  "
```

Authentication possible lemma checks whether both hosts can authenticate the other party through the full run of the protocol. As the figure 6.27 points out, hosts can authenticate each other without problem.

Listing 6.28: Execution results are positive for authentication possible lemma of One Time Password Scheme

```
lemma Authentication_Possible:
  exists-trace
  "(All x y #i. (Eq( x, y ) @ #i) ==> (x = y)) and
        (Ex U S V1 A1 F2 #j.
          (!S_Authenticates_U( S, U, V1, A1, F2 ) @ #j) and
          (Ex R1 F1 hRF #k.
            !U_Authenticates_S( U, S, R1, F1, hRF ) @ #k))"
simplify
solve( Server_Reg_2( ID, h(A1), h(<A1, F2>) ) #j )
  case Server_Reg_2
  solve( !Credentials( U, S ) #j )
    case Register_US
    solve( User_Auth_1( A3, F3 ) #k )
      case User_Auth_1
      solve( !Credentials( $U, $S ) #k )
        case Register_US
        solve( User_Reg_2_FQ( F1, Q2 ) #k )
          case User_Reg_2
          solve( !KU( h(<h(h(<~ID, ~PW, ~F0>)), V2R>) ) @ #vk.6 )
            case User_Auth_1
            solve( !KU( h(<R1.1, h(h(<~ID.2, ~PW.1, ~F0.1>))>)
                   ) @ #vk.9 )
              case Server_Auth_1
              solve( !KU( senc(h(h(<~ID.2, h(<~PW.1, ~Rm1.2>),
                                     h(h(<~ID.2, ~PW.1, ~F0.1>))>)),
                               h(h(<~ID.2, ~PW.1, ~F0.1>)))
                     ) @ #vk.11 )
                case User_Auth_1
                solve( !KU( h(<h(h(<~ID.2, ~PW.1, ~F0.1>)), V2R>)
                       ) @ #vk.16 )
```

```
case User_Auth_1
solve( !KU( ~R1.1 ) @ #vk.12 )
  case Server_Auth_1
  solve( !KU( senc(h(h(<~ID, h(<~PW, ~Rm1>),
                          h(h(<~ID, ~PW, ~F0>))>)),
                      h(h(<~ID, ~PW, ~F0>)))
         ) @ #vk.11 )
    case User_Auth_1
    solve( !KU( senc(h(<~ID, ~PW, ~F0>),
                     h(h(<~ID, h(<~PW, ~Rm1>),
                          h(h(<~ID, ~PW, ~F0>))>)))
           ) @ #vk.13 )
      case User_Auth_1
      solve( !KU( h(<
                     h(<~ID, h(<~PW, ~Rm1>), h(h(<~ID, ~PW, ~F0>))
                       >),
                     h(h(<~ID, h(<h(<~PW, ~Rm1>), ~R0>),
                          h(h(<~ID, h(<~PW, ~Rm1>),
                               h(h(<~ID, ~PW, ~F0>))>))
                       >))
                   >)
             ) @ #vk.14 )
        case User_Auth_1
        solve( !KU( senc(h(<~ID.2, ~PW.1, ~F0.1>),
                         h(h(<~ID.2, h(<~PW.1, ~Rm1.2>),
                              h(h(<~ID.2, ~PW.1, ~F0.1>))>)))
               ) @ #vk.15 )
          case User_Auth_1
          solve( !KU( h(<
                         h(<~ID.2, h(<~PW.1, ~Rm1.2>),
                            h(h(<~ID.2, ~PW.1, ~F0.1>))>),
                         h(h(<~ID.2,
                              h(<h(<~PW.1, ~Rm1.2>), ~R0.2>),
                              h(h(<~ID.2, h(<~PW.1, ~Rm1.2>),
                                   h(h(<~ID.2, ~PW.1, ~F0.1>))>))
                            >))
                       >)
                 ) @ #vk.16 )
            case User_Auth_1
            SOLVED // trace found
          qed
```

Listing 6.29: Implementation of registration possible lemma of One Time Password Scheme

```
lemma Registration_Possible:
  exists-trace
  "Ex ID #i. Registration(ID) @i "
```

Listing 6.30: Execution results are positive for registration possible lemma of One Time Password Scheme

```
lemma Registration_Possible:
  exists-trace "Ex ID #i. Registration( ID ) @ #i"
simplify
solve( User_Reg_2_S( F1, V1 ) #i )
  case User_Reg_2
  solve( Server_Reg_1( ID.1, R0, Rm1.1, F0.1 ) #i )
    case Server_Reg_1
    SOLVED // trace found
  qed
qed
```

Registration possible lemma exists to check whether is it possible to make a successful registration through the registration phase of the protocol. It can be inferred from the figure 6.30 that registration is possible.

7 Conclusion and Future Work

In conclusion, throughout the project stage, so many various computer security terms and concepts are learned. I got familiar with the automatic protocol verification tools AVISPA and Tamarin Prover. AVISPA is only used to read and understand the different implementations of the security protocols. Implementations are done on the Tamarin Prover and due to this reason, I have good knowledge to the Tamarin Prover tool. CRAM-MD5, APOP, MS-CHAPv2, Kerberos v5, One Time Password Scheme [16] and basic TLS protocol are learned and all of them except TLS are implemented on Tamarin Prover.

CRAM-MD5, APOP, MS-CHAPv2 and Kerberos v5 are previously verified through automatic security protocol verification tool AVISPA. All of them are found to be secure. My results also match those results and since they match, those protocols could be used in the internet transmission without doubt. However, there is an exception in my outcomes. Results of my implementation of Kerberos protocol states that it is possible to create a session but secrecy and authentication lemmas fails to ensure the intended properties. The reason for this could be the possible mistakes in the implementation of the protocol or flaws of the protocols itself.

Computer security is vast and fun computer science branch but my topic is not suitable for a graduation project. I did not have much knowledge about computer security concepts before starting the project and along with learning all those concepts, getting used to the tools and understanding automatic theorem verification that contains so much mathematical logic was not easy task. Students that will study the same topic must take these into consideration and choose graduation project topic carefully.

References

[1] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," 2017.

[2] AVISPA project. Accessed: 10-05-2017. [Online]. Available: http://web.archive.org/web/20170605125914/http://www.avispa-project.org/

[3] T. Nipkow. Theorem proving with isabelle/hol an intensive course. [Online]. Available: http://isabelle.in.tum.de/coursematerial/PSV2009-1/

[4] Sharemind. Sharemind secrec compiler and analyzer. [Online]. Available: https://github.com/sharemind-sdk/secrec

[5] C. Hriţcu, "F*: From program verification system to proof assistant."

[6] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed. Pearson, 2011.

[7] M. Rouse. What is confidentiality, integrity, and availability (CIA triad)? Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130646/http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA

[8] T. Chia. (2012) Confidentiality, integrity, availability: The three components of the cia triad. Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130733/http://security.blogoverflow.com/2012/08/confidentiality-integrity-availability-the-three-components-of-the-cia-triad/

[9] M. Rouse. (2005) Data integrity. Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130806/http://searchdatacenter.techtarget.com/definition/integrity

[10] D. Miessler. (2005) Security: Identification, authentication, and authorization. Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130910/https://danielmiessler.com/blog/security-identification-authentication-and-authorization/

[11] T. Leek. (2012) Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130929/https://security.stackexchange.com/questions/10933/difference-between-authentication-and-identification-crypto-and-security-perspe

[12] M. Rouse. (2014) What is authentication factor? Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605130956/http://searchsecurity.techtarget.com/definition/authentication-factor

[13] B. University. Understanding authentication, authorization, and encryption. Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605131032/https://www.bu.edu/tech/about/security-resources/bestpractice/auth/

[14] Cryptography hash functions. Accessed: 15-05-2017. [Online]. Available: http://web.archive.org/web/20170605131106/https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm

[15] M. Peignon. (2017) Accessed: 15-05-2017. [Online]. Available: https://www.quora.com/Why-are-true-random-number-generators-more-secure-than-pseudo-random-number-generators

[16] R. Isawa and M. Morii, "One time password scheme to solve stolen verifier problem," *Information Processing Society of Japan and The Institute of Electronics, Information and Communication Engineers*, 2011.

[17] The Tamarin Team, "Tamarin prover manual," 2016.

[18] AVISPA. CRAM-MD5. Accessed: 18-05-2017. [Online]. Available: http://web.archive.org/web/20170605131211/http://www.avispa-project.org/library/CRAM-MD5.html

[19] (2013) Accessed: 20-05-2017. [Online]. Available: http://web.archive.org/web/20170605131334/https://security.stackexchange.com/questions/36117/how-secure-is-using-cram-md5-for-email-authentication-when-not-using-an-ssl-con

[20] Authenticated post office protocol (APOP). Accessed: 20-05-2017. [Online]. Available: http://web.archive.org/web/20170605131502/https://www.techopedia.com/definition/1632/authenticated-post-office-protocol-apop

[21] (2016) APOP (authenticated post office protocol). Accessed: 20-05-2017. [Online]. Available: http://web.archive.org/web/20170605131416/http://www.tech-faq.com/apop.html

[22] Microsoft Corporation. (2000) Microsoft PPP CHAP extensions, version 2. Accessed: 20-05-2017. [Online]. Available: http://web.archive.org/web/20170605131531/https://tools.ietf.org/html/rfc2759

[23] MIT KERBEROS. (2006) Kerberos version 4 end of life announcement. Accessed: 20-05-2017. [Online]. Available: http://web.archive.org/web/20170605131546/https://web.mit.edu/kerberos/krb4-end-of-life.html

[24] AVISPA. APOP. Accessed: 18-05-2017. [Online]. Available: http://web.archive.org/web/20170605131226/http://www.avispa-project.org/library/apop.html

[25] ——. CHAP. Accessed: 18-05-2017. [Online]. Available: http://web.archive.org/web/20170605131245/http://www.avispa-project.org/library/CHAPv2.html

[26] Explain like i'm 5: Kerberos. Accessed: 24-05-2017. [Online]. Available: http://web.archive.org/web/20170605131600/http://www.roguelynn.com/words/explain-like-im-5-kerberos/

[27] AVISPA. KERBEROS v5. Accessed: 18-05-2017. [Online]. Available: http://web.archive.org/web/20170605131320/http://www.avispa-project.org/library/Kerb-basic.html

[28] S. Özgür. (2017) CRAM-MD5 tamarin prover model. [Online]. Available: https://github.com/Klypnos/tamarin-grad/blob/master/CramMD5.spthy

[29] ——. (2017) MS-CHAPv2 tamarin prover model. [Online]. Available: https://github.com/Klypnos/tamarin-grad/blob/master/CHAPv2.spthy

[30] ——. (2017) APOP tamarin prover model. [Online]. Available: https://github.com/Klypnos/tamarin-grad/blob/master/APOP.spthy

[31] ——. (2017) One time password scheme tamarin prover model. [Online]. Available: https://github.com/Klypnos/tamarin-grad/blob/master/OTP.spthy

[32] ——. (2017) Kerberos v5 tamarin prover model. [Online]. Available: https://github.com/Klypnos/tamarin-grad/blob/master/KerberosBasic.spthy

8 Appendix

Listing 8.1: Implementation of CRAM-MD5 protocol [28]

```
theory CramMD5
begin

/*
Alice-Bob Notation:
 1. A -> S: A
 2. S -> A: Ns.T.S
 3. A -> S: F(SK.T)
 where
   Ns is a nonce generated by the server;
   T is a timestamp (currently abstracted with a nonce)
   SK is the shared key between A and S
   F is a cryptographic hash function (MD5 in practice, but this is
    unimportant for our purposes). The use of F
    is intended to ensure that only a digest of the shared
    key is transmitted, with T assuring freshness of the
    generated hash value.
*/

functions: h/1

rule Register_SK:
  [ Fr(~SK) ]
  -->
  [ !Shared_Key($A,$S,~SK) ]

rule Reveal_SK:
  [ !Shared_Key(A, S, SK)]
  --[ SK_Reveal(A, S) ]->
  [ Out(SK) ]

// 1. A -> S: A
rule Client_1:
  [ !Shared_Key(A, S, K) ]
  -->
  [ Client_1(A),
    Out(A) ]

rule Server_1:
  [ In(A),
    Fr(~T),
    Fr(~N),
    !Shared_Key(A, S, K)
  ]  // 2. S -> A: Ns.T.S
  -->
  [ Server_1(~T),
  Out( <~N, ~T, S> ) ]


rule Client_2:
  [ In( < N, T, S > ),
    !Shared_Key(A, S, SK),
    Client_1(A)
  ]  //3. A -> S: F(SK.T)
  -->
  [
    Out( h( < SK,T > ) )
  ]

rule Server_2:
  [
    !Shared_Key(A, S, SK),
    In( hashed ),
    Server_1( T )
  ]
  --[
    Eq( hashed, h(SK,T) ),
    AnsweredRequest(S, SK),
```

```
    Session( A, S, SK, T )
  ]->
  []

axiom Equality_Checks_Succeed: "All x y #i. Eq(x,y) @ i ==> x = y "

lemma SK_secrecy:
  "
    not(
    Ex A S SK T #i #j.
      Session( A, S, SK, T ) @ #i
    & K(SK) @ #j
    & not( Ex #r. SK_Reveal(A, S) @ #r)
    )
  "

lemma Client_Authentication:
  "
  (  All A S SK T #i. Session( A, S, SK, T ) @ #i
    ==>
    (
      (( Ex #a. AnsweredRequest(S,SK) @ a )
         & (All #j. Session(A,S,SK,T) @ #j ==> #i = #j) )
      | ( Ex #r. SK_Reveal(A, S) @ r & r < i )
    )
  )
  "

/* Consistency check: ensure that session-keys can be setup between honest
 * agents. */
lemma session_key_setup_possible:
  exists-trace
  " /* There is a trace satisfying all equality checks */
   (All x y #i. Eq(x,y) @ i ==> x = y)
  &  /* Session keys have been setup */
   (Ex A S SK T #k.  Session(A, S, SK, T) @ k
    /* without having performed a long-term key reveal. */
    & not (Ex #r. SK_Reveal(A, S) @ r)
    )
   "
end
```

Listing 8.2: Implementation of MS-CHAPv2 protocol [29]

```
theory CHAPv2
begin

/*
We assume that the server B and client A share password k(A,B) in advance. The server
    and client generate nonces Nb and Na, respectively.
  1. A -> B : A
  2. B -> A : Nb
  3. A -> B : Na,H(k(A,B),(Na,Nb,A))
  4. B -> A : H(k(A,B),Na)

*/


/*
Not sending k(A,B) at 3rd step, violates only the honest_key_setup lemma. Is this
    expected?
*/

// lowe
functions: h/1

rule Register_SK:
  [ Fr(~SK_AB)]
  -->
  [ !Shared_Key($A, $B, ~SK_AB) ]

rule Reveal_SK:
  [ !Shared_Key(A, B, SK_AB)]
```

```
  --[ SK_Reveal(A,B) ]->
  [ Out(SK_AB) ]

rule Client_1: // Client is A, Server is B
  [ !Shared_Key(A, B, SK_AB) ]
  -->
  [ Out(A),
    Client_1(A, B, SK_AB)
   ]

rule Server_1:
  [ In(A),
    Fr(~Nb)
  ]
  -->
  [ Out(~Nb),
    Server_1(~Nb)
   ]

rule Client_2:
  [ In(Nb),
    Fr(~Na),
    Client_1(A,B,SK_AB)
  ]
  --[ Client_Answered(A, B, Nb, SK_AB) ]->
  [
    Out(<~Na,h(<SK_AB,~Na,Nb,A>)>),
    Client_2(~Na,Nb,A,B,SK_AB)
  ]
rule Server_2:
  [ In( hashed ),
  //In(<Na,h(<SK_AB,Na,Nb,A>)>),
    !Shared_Key(A, B, SK_AB),
    Server_1(Nb)
  ]
  --[ Eq( snd(hashed), h( <SK_AB, fst(hashed), Nb, A> ) ),
    Auth_Nb( A, B, SK_AB, Nb),
    Server_Answered(A, B, fst(hashed), SK_AB)
  //Server side auth complete
  ]->
  [
   Out(h(<fst(hashed), SK_AB>))
  ]

rule Client_3:
  [ In( hashed ),
  //In(h(<SK_AB,Na>)),
    Client_2(Na,Nb,A,B,SK_AB)]
  --[ Eq( hashed, h( <Na, SK_AB> ) ), // client side auth complete
    Auth_Na( A, B, SK_AB, Na),
    Session(Na,Nb,A,B,SK_AB)
    ]->
  []

axiom Equality_Checks_Succeed1: "All x y #i. Eq(x,y) @ i ==> x = y "
//axiom Equality_Checks_Succeed2: "All x y #i. Eq2(x,y) @ i ==> x = y "

lemma SK_secrecy:
  " /* It cannot be that a */
  not(
    Ex Na Nb A B SK_AB #i #j.
      Session( Na, Nb, A, B, SK_AB ) @ #i
    & K(SK_AB) @ #j
    & not( Ex #r. SK_Reveal(A,B) @r)
  )
  "
// Eq statements shouldnt be in lemmas ( except the honest setup lemma ) ?????
lemma Auth_on_Na:
  "
  ( All A B SK_AB Na #i. Auth_Na(A, B, SK_AB, Na) @ #i
    ==>
    ( (Ex #j. SK_Reveal(A, B) @ j & j < i)
      | ( Ex #r. Server_Answered(A, B, Na, SK_AB) @r
```

```
        &  (All #k. Auth_Na(A, B, SK_AB, Na) @ #k ==> #i = #k   )
        )
    )
  )
  "

lemma Auth_on_Nb:
  "
  ( All A B SK_AB Nb #i. Auth_Nb  (A, B, SK_AB, Nb) @ #i
    ==>
    (
      (Ex #j. SK_Reveal(A, B) @ j & j < i)
      | (Ex #r. Client_Answered(A, B, Nb, SK_AB) @r
      & (All #k. Auth_Nb  (A, B, SK_AB, Nb) @ #k ==> #i = #k))
    )
  )
  "

lemma session_key_setup_possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @ i ==> x = y)
  &
  ( Ex Na Nb A B SK_AB #r. Session( Na, Nb, A, B, SK_AB ) @ #r
    & not( Ex #j. SK_Reveal(A, B) @j )
  )
  "
end
```

Listing 8.3: Implementation of APOP [30]

```
theory APOP
begin

/*
S -> C : Hello.Timestamp
 C -> S : C.MD5(Timestamp.K_CS)
 S -> C : Success
*/

builtins: hashing
//functions: h/1

rule Register_SK:
  [ Fr(~SK)]
  -->
  [ !Shared_Key($S, $C, ~SK) ]

rule Reveal_SK:
  [ !Shared_Key(S, C, SK)]
  --[ SK_Reveal(S, C) ]->
  [ Out(SK) ]

rule Server_1:
  [ Fr(~T) ]
  -->
  [ Out(<'Hello', ~T>),
    Server_1(~T)
  ]

rule Client_1:
  [ In(<'Hello', T>),
    !Shared_Key(S, C, SK)
  ]
  --[ AnswerReq(S, C, SK, T)]->
  [ Out(<C,h(<T, SK>)>)]

rule Server_2:
  [ In( hashed ),
    !Shared_Key(S, C, SK),
    Server_1(T)
  ]
  --[ Eq( snd(hashed), h(<T, SK>) ),
```

```
    Eq( fst(hashed), C),
    AuthDone(S, C, SK, T)]->
  [ Out('Success') ]

rule Client_2:
  [ In(x),
    !Shared_Key(S, C, SK),
    Server_1(T) ]
  --[ Eq(x,'Success'),
    Successful(S, C, SK, T) ]->
  []

axiom Equality_Checks_Succeed: "All x y #i. Eq(x,y) @ i ==> x = y "

lemma SK_secrecy:
  " /* It cannot be that a */
  not(
    Ex S C SK T #i #j.
      Successful( S, C, SK, T ) @ #i
    & K(SK) @ #j
    & not( Ex #r. SK_Reveal(S, C) @r)
  )
  "

lemma Authentication:
  "
    ( All S C SK T #i. AuthDone( S, C, SK, T ) @ #i
      ==>
       ( (Ex #r. SK_Reveal(S, C) @ r & r < i)
        | (Ex #l. AnswerReq(S, C, SK, T) @l
        & (All #j. AuthDone( S, C, SK, T ) @ #j ==>  #i = #j )
        )
       )
    )
  "

lemma session_key_setup_possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @ i ==> x = y)
  &
  (Ex S C SK T #j. Successful(S, C, SK, T) @ #j
    & not( Ex #r. SK_Reveal(S, C) @r )
  )
  "
end
```

Listing 8.4: Implementation of One Time Password Scheme [31]

```
theory OTP
begin

builtins: symmetric-encryption, hashing


/* Registration */

//functions: xor/2

rule Register_PW:
  [ Fr(~PW) ]
  -->
  [ Pass($U, ~PW) ]

rule Reveal_PW:
  [ Pass($U, ~PW) ]
  --[ PW_Reveal($U)]->
  [ Out(~PW) ]

rule Register_US:
  [ ]
  -->
  [ !Credentials($U, $S) ]
```

```
/* REGISTRATION PHASE START */
rule User_Reg_1: // Does it matter where I generate a fresh key? Here I dont use PW in
     this step,
  [   Fr(~ID), // but PW is normally already generated at this step.
    Pass( $U, ~PW)
  ]
  -->
  [   User_Reg_1(~ID, ~PW),
    User_Reg_1_S(~ID)
  ]

rule Server_Reg_1:
  [   Fr(~R0),
    Fr(~Rm1),
    Fr(~F0),
    User_Reg_1_S(ID)
  ]
  -->
  [
    Server_Reg_1(ID, ~R0, ~Rm1, ~F0),
    Server_Reg_1_U(~R0, ~Rm1, ~F0)
  ]
rule User_Reg_2:
  let
    A1 = h(<ID,PW,F0>)
    F1 = h(A1)
    Q1 = h(<PW,Rm1>)
    A2 = h(<ID,Q1,F1>)
    F2 = h(A2)
    Q2 = h(<Q1,R0>)
    V1 = h(<A1,F2>)
  in
  [
    User_Reg_1(ID, PW),
    Server_Reg_1_U(R0, Rm1, F0)
  ]
  -->
  [
    User_Reg_2(ID,Q2,A1,F1,A2,F2),
    User_Reg_2_FQ(F1,Q2),
    User_Reg_2_S(F1,V1)
  ]

rule Server_Reg_2:
  [   User_Reg_2_S(F1,V1),
    Server_Reg_1(ID, R0, Rm1, F0)
  ]
  --[
    Registration(ID)
    ]->
  [
    Server_Reg_2(ID,F1,V1)
  ]

/* REGISTRATION PHASE END */

/* AUTHENTICATION PHASE START */

rule User_Auth_1:
  let
    A3  = h(<ID,Q2,F2>)
    F3  = h(A3)
    Fc  =  senc(F2,F1) // xor workaround
    AFc =  senc(A1,F2) //xor workaround
    V2  = h(<A2,F3>)
    FVc = h(<F1,V2>)
  in
  [
    User_Reg_2(ID, Q2, A1, F1, A2, F2)
  ]
  -->
  [
```

```
    User_Auth_1(A3,F3),
    Out(<Fc, AFc, V2, FVc>)
  ]

rule Server_Auth_1:
  let
    calc_F2 = sdec(FcR, F1)
    calc_A1 = sdec(AFcR, calc_F2)
    calc_F1 = h(calc_A1)
    calc_V1 = h(<calc_A1,calc_F2>)
    calc_FVc = h(<F1, V2R>)
  in
  [   Fr(~R1),
    In(<FcR, AFcR, V2R, FVcR>),
    Server_Reg_2(ID, F1, V1),
    !Credentials(U, S)
    //seperate this maybe
  ]
  --[
  //  Answer_Req(U, S, )
    Eq( F1, calc_F1),
    //S IS CONVINCED THAT F1 F2 A1 ARE NOT MODIFIED IF ABOVE EQ CHECKS SUCCEED
    Eq( V1, calc_V1),
    // S AUTHENTICATES U
    !S_Authenticates_U(S,U, calc_V1, calc_A1, calc_F2),
    Eq( FVcR, calc_FVc )
  ]->
  [
    Server_Auth_1(calc_F2,V2R),
    Out( <~R1,h(<~R1,F1>)> )
  ]

rule User_Auth_2:
  let
    calc_hRF = h(<R1, F1>)
    Q3 = h(<Q2, R1>)
  in
  [
    In(<R1, hRF>),
    User_Auth_1(A3,F3),
    !Credentials(U, S),
    User_Reg_2_FQ(F1,Q2)
  ]
  --[ Eq( hRF, calc_hRF ),
    !U_Authenticates_S(U, S, R1, F1, hRF)]->
  [
    User_Auth_2(Q3, A3, F3)
  ]

/* AUTHENTICATION PHASE END */
axiom Equality_Checks_Succeed: "All x y #i. Eq(x,y) @ i ==> x = y "

lemma Authentication_Possible:
  exists-trace
  "
  (All x y #i. Eq(x,y) @i ==> x = y)
  &
  ( Ex U S V1 A1 F2 #j. !S_Authenticates_U(S,U,V1,A1,F2) @j
    & Ex R1 F1 hRF #k. !U_Authenticates_S(U,S,R1,F1,hRF) @k )
  "

lemma Registration_Possible:
  exists-trace
  "Ex ID #i. Registration(ID) @i "

end
```

Listing 8.5: Implementation of Kerberos v5 protocol [32]

```
theory KerberosBasic
begin
```

```
builtins: hashing, symmetric-encryption

/*
C: Client
 A: Authentication Server
 G: Ticket Granting Server
 S: Server (that the client wants to talk to)

 K_AB: key shared or intended to be shared between A and B
      Initially shared: K_CA, K_AG, K_GS
      Established during protocol: K_CG, K_CS


 All things marked * are timestamp-related and will be simply replaced
 with fresh text.


 Macros:
 Ticket_1 := { C,G, K_CG, Tstart*, Texpire* }K_AG
 Ticket_2 := { C,S, K_CS, Tstart2*, Texpire2* }K_GS


 1. C -> A : C,G,Lifetime_1*,N_1
 2. A -> C : C, Ticket_1, { G, K_CG, Tstart*, Texpire*, N_1 }K_CA


 3. C -> G : S,Lifetime_2*,N_2,Ticket_1, { C,T* }K_CG
 4. G -> C : C, Ticket_2, { S, K_CS, Tstart2*, Texpire2*, N_2 }K_CG


 5. C -> S : Ticket_2, { C, T2* }K_CS
 6. S -> C : { T2* }K_CS

*/
rule Register_SK:
  [ Fr(~Key_XY) ]
  -->
  [ !Shared($X, $Y, ~Key_XY)]

rule Reveal_SK:
  [ !Shared(X,Y,Key_XY)]
  --[ SK_Reveal(X,Y)]->
  [ Out(Key_XY)]

rule Parties:
  []
  -->
  [ !Parties($A,$C,$G,$S) ]

rule Client_1: //C
  [ Fr(~LifeTime_1),
  Fr(~N_1),
  !Parties(A,C,G,S)]
  -->
  [ Out(<C,G,~LifeTime_1, ~N_1>) ]

rule AuthServ_1://A
  let
  Ticket_1 = senc(<C, G, ~Key_CG, ~Tstart, ~Texpire> ,Key_AG)
  in
  [ Fr(~Tstart),
  Fr(~Texpire),
  Fr(~Key_CG),
  !Shared(A, G, Key_AG),
  !Shared(C, A, Key_CA),
  In(<C, G, LifeTime_1, N_1>)
  ]
  -->
  [
  Out(<C, Ticket_1, senc(<G, ~Key_CG, ~Tstart, ~Texpire, N_1>,Key_CA) >)
  ]

rule Client_2://C
```

```
  [ Fr(~LifeTime_2),
  Fr(~N_2),
  Fr(~T),
  !Parties(A, C, G, S),
  In(<C, Ticket_1, senc(<G, Key_CG, Tstart, Texpire, N_1>,Key_CA) >)
  ]
  --[ Answer_CG(C, G, A, Key_CG) ]->
  [ Out(<S, ~LifeTime_2, ~N_2, Ticket_1, senc(<C, ~T>,Key_CG) >) ]

rule GrantingServ_1://G
  let
  Ticket_2 = senc(<C, S, ~Key_CS, ~Tstart2, ~Texpire2>,Key_GS)
  in
  [ Fr(~Tstart2),
  Fr(~Texpire2),
  Fr(~Key_CS),
  !Parties(A, C, G, S),
  !Shared(G, S, Key_GS),
  In(< S, LifeTime_2, N_2, Ticket_1, senc(<C, ~T>,Key_CG) >)]
  --[ SessKeyCG(C, G, A, Key_CG)]->
  [ Out(< C, Ticket_2, senc(<S, ~Key_CS, ~Tstart2, ~Texpire2, N_2>,Key_CG) >) ]
rule Client_3://C
  [ Fr(~T2),
  In(< C, Ticket_2, senc(<S, Key_CS, Tstart2, Texpire2, N_2>,Key_CG) >)
  ]
  -->
  [ Out(< Ticket_2, senc(<C, ~T2>,Key_CS)>)]
rule Server_1: //S
  [
  In(< Ticket_2, senc(<C, T2> ,Key_CS) >)
  ]
  -->
  [
  Out(<senc(T2,Key_CS)>)
  ]
rule Client_4: //C
  [
  In(<senc(T2,Key_CS)>),
  !Parties(A, C, G, S)
  ]
  --[ Session(C, S, <'client', Key_CS >)]->
  []

axiom Equality_Checks_Succeed: "All x y #i. Eq(x,y) @ i ==> x = y "

lemma Session_Key_Setup_Possible:
  exists-trace
  "
  Ex C S params #j. Session(C, S, params) @j
   & not( Ex A #r. SK_Reveal(C,A) @r )
   & not( Ex A G #s. SK_Reveal(A,G) @s )
   & not( Ex G #t. SK_Reveal(G,S) @t )
  "

lemma CG_secrecy:
  " /* It cannot be that a */
  not(
    Ex C G A k #i #j.
      /* client has set up a session key 'k' with a server 'S' */
      SessKeyCG(C, G, A, k) @ #i
      /* and the adversary knows 'k' */
    & K(k) @ #j
      /* without having performed a long-term key reveal on 'S'. */
    & not(Ex #r. SK_Reveal(A, G) @ r) | (Ex #l. SK_Reveal(C, A) @ l) )
  )
  "

  lemma CG_weak_auth:
    " /* For all session keys 'k' setup by clients with a server 'S' */
      ( All C G A k #i. SessKeyCG(C, G, A, k) @ #i
      ==>
        /* there is a server that answered the request */
        ( (Ex #a. Answer_CG(C, G, A, k) @ a)
```

```
        /* or the adversary performed a long-term key reveal on 'S'
        before the key was setup. */
        | (Ex #r. SK_Reveal(A, G) @ r & r < i)
        | (Ex #l. SK_Reveal(C, A) @ l & l < i)
        )
    )
  "

//Initially shared: K_CA, K_AG, K_GS
end
```