Challenges of Trust Establishment

Kirill Sedow
Technical University of Munich
Munich, Germany
kirill.sedow@tum.de

***Abstract*—Trust is an important means of establishing security in distributed networks. In order for a public key infrastructure to exist, trust needs to be formalized and computed. Our work discusses different approaches of trust formalization and provides an overview of different trust establishment techniques. While providing theory, we implement a concrete trust computation algorithm to illustrate the theory.**

***Index Terms*—public key infrastructure, web of trust, trust, security services**

I. Introduction

When browsing the Internet many processes stay hidden from the user. By calling the page `https://google.com` in a browser, a web page is displayed on a user's screen. In particular, the browser knows that the given webpage belongs to Google. But how can we be sure that no other user is sitting in a room next to us and sends us this webpage in the name of Google?

A simple look into `traceroute` [16] will reveal that the search query takes many steps before reaching one of Google's servers. And somehow our computer is sure that the server indeed belongs to Google.

Another example can be found in digital marketing. Imagine that we want to take a EUR20,000 bank loan. We open Check24 and enter our query. After that, we can look for a suitable institution and contact for further discussions.

Two points are interesting here: first, we are sure that we are browsing the Check24 page (and not some other page acting in the name of Check24). Secondly, we believe that the institutions offered through Check24 are those who they claim to be. How can we be sure that the website in question, and the loan, are genuine?

The trust and security establishment in distributed networks is a complex matter, as intuitive understanding of trust and security has to be formalized and adapted into a digital world. If two entities in a network wish to establish a connection, we usually say that they **establish a channel** to each other. If this channel is secure, we speak about a **secure channel**.

However, no channel is secure by default: communication in the network happens over the wires that are used by many communication parties. Different kinds of multiplexing are used, so that these communication parties can share the same channel at the same time. As a consequence, communication does not happen in isolation, but can be measured, tracked and redirected by third parties.

So, it is natural to see every communication channel as an insecure channel. However, applying cryptography we are able to use the insecure channel as a secure channel. Cryptography does not create security [15], but enables the creation of a time-dependent secure channel which can be used at a later point in time in order to fulfill some **security services**.

Trust is another matter which is different from security. Intuitively we can say that we trust an entity x if entity x behaves as expected, and if entity x behaves as expected during the whole history of our communication. We can trust somebody without having a secure channel, and we can have a secure channel with a person we do not even know. Furthermore, there is also the trust between network entities can be defined as a measure of the capability to resist any attempt at of malicious manipulations [1], while moreover, trust can also be said to be when we believe that the public key of x is authentic.

Intuitive trust can be demonstrated in an online shopping example; we are consistently sure that if we made a payment successfully once, then the next payment should proceed successfully, as well. Thus, trust is formed not at the moment of interaction, but also has a history. This history can be logged and tracked in order to make precise formalizations about what the trust is, or how the trust should be measured. The consistency in performing certain services all the time without leaking our data to third parties leads us to the term of **confidentiality**.

Network trust, a second aspect is when entity x is trustworthy if no entity y can make x to behave differently from expected. This shows a different aspect of trust: the system we are interacting with must not only provide the required services honestly and in an expectedly way, but must also be protected against any harmful influence aside. Hence, protection is another prerequisite for trust. We want to expect some **integrity** from the service we are using.

A third aspect of trust is when we say that we trust x, if we believe that the public key of x is authentic, and hence belongs to x (and not some other y that we do not know). This is the definition of trust we are going to focus on. It is not only important to provide services consistently, but also to ensure that no other service in the world can perform the same services in another's name. These thoughts lead us to the terms of **authenticity** and **non-repudiation**.

In particular, we are going to have a closer look at the questions mentioned before, and show how different security services can accomplish our expectations in establishing trustworthy relationships. We are going to go through the techniques that will allow us to formalize trust and make our discussion more complete and precise.

The rest of the paper is structured as follows: section II mentions related works, section III provides the background necessary for our discussion, section IV defines a public key infrastructure (PKI) in terms of our background knowledge, section V gives an overview of different kinds of PKI architectures, section VI gives a model and an algorithm for trust computations, and section VII provides the evaluation of our work.

This paper discusses different background formalization techniques and reveals which extensive work has to be done to make secure communications possible to ensure security while browsing the Internet.

II. Related Work

In a technical sense, trust exists as long as computers and networks do. This results in the fact that an enormous amount of work has been done in this direction during the whole history of computer science.

In particular, a great amount of work introduces the public key infrastructure. We are going to reference the works in [19] and [6], as the techniques described there are quite common. Being a generalization of trust relationships, the **Web of Trust** is introduced in a number of works. Blockchain-based PKI-s [13], [10] are other variations of a trust infrastructure. Such works make clear that almost every new technology can be taken and applied in order to establish trust and security.

We believe that the cornerstones of the trust formalization are done in [3] and [5]. U. Maurer provides a comprehensive guide on modeling PKI infrastructures, while these works can be considered as general as possible. We will cover the main ideas of these papers later in our work. The work in [2] provides a concrete simplification of the general Maurer's model and introduces a heuristic approach for the computation of trust.

Depending on the area of application, different algorithms for trust computation exist [20], [21]. There is no common approach suitable for all situations, as a trust level for a bank account should differ from a trust level in social networks. One consequence is that different trust calculations can differ, but be valid in their own area of application. An example could be the work in [22], which analyzes the trust and recommendation metrics of CouchSurfing, a network that boomed 10-15 years ago and provided an ability to find a place to stay, when traveling to another city.

Recent publications show an interesting application of game theory [1] iinto the calculation process. Briefly, imagine two actors, Alice and Bob, having different strategies towards the routing of packages in a network, combined with their trust in the routing connections. We can capture their trust values in a **payoff matrix**:

$$\begin{bmatrix} & B_1 & B_2 & B_3 \\ A_1 & 0.8 & 0.7 & 0.55 \\ A_2 & 0.3 & 0.9 & 0.95 \\ A_3 & 0.4 & 0.55 & 0.9 \end{bmatrix}$$

This matrix should be read as follows: Alice (A) has three different routing schemes: A_1, A_2 and A_3. When choosing a routing scheme, Alice gives Bob (B) a chance to answer either with B_1, B_2 or B_3. If Alice took the strategy A_1 and Bob answered with B_2, then the payoff of Alice is 0.7. Hence, we can say that Alice trusts Bob with respect to this routing scheme with a value of 70%.

The next step is to introduce strategies. If Alice **plays with caution**, she will take a look at the strategies, choose a minimum value in each row, and then choose a maximum value over the minima. In our example, Alice would choose 0.55 in row A_1, 0.3 in row A_2 and 0.4 in row A_3. After that, she takes $\max\{0.55, 0.3, 0.4\} = 0.55$, hence deciding herself for the strategy A_1. Bob can answer with a different strategy, while his action will depend on the previous action of Alice. Fulfilling the model, it is possible to assign dynamic trust values to all different connections and update the trust values according to the reactions of the opponent.

Another interesting alternative makes use of fuzzy logic in order to determine the trust value [23], [24]. In fuzzy logic, we can introduce **semantic variables**. For example, we can say that $trust$ is a semantic variable which can take the values $high$, mid, or low. We can define a level of security and recommendations as other semantic variables. This allows us to introduce a set of rules which can be used in order to determine the trust value. It can then be said that if the $security$ has a value $almostsecure$ and the recommendation is $positive$, then the resulting $trust$ value is $high$. Further transformations (fuzzification and defuzzification) allow us to map semantic variables to numerical values, which can be later used in computation.

The main dilemma in the works about trust lies in where to place the focus. A model can either have an expressional theoretical power, or provide a measurable and effective calculation algorithm. The work in [3] provides a theoretically powerful model, but introduces an exponential algorithm, which is not applicable for a large number of input parameters. The work in [2] simplifies the model of [3], which leads to a loss of generality. However, the presented algorithm is faster and can be applied in a practical scenario.

This comparison demonstrates that efficiency and generality are barely combined together inside one approach. As a consequence, works in the given area tend to provide either a concrete implementation algorithm (which can be used directly out-of-the-box), or introduce a general and verifiable model (which cannot be used out-of-the-box due to its complexity).

III. Preliminaries

This section introduces the background theory, which is necessary for our discussion. When speaking about two communication parties, we usually call them Alice and Bob, - as it is useful to call them this way [27]. Alice and Bob are usually *good* actors. In contrast, Eve is considered to be a *bad* actor trying to obtain the secret information shared by Alice and Bob. We will also use Charlie, who is as good as Alice and Bob.

A. Security services

Security of a computer system is measured by the ability of the system to provide certain **security services**. The most common security services are integrity, confidentiality, authenticity and non-repudiation.

- We speak about **data integrity**, if the system is able to prevent an unauthorized or accidental modification of data. All the data must be maintained and modified only by authorized people. In a more common sense, data integrity should also be verified after a transition through a noisy channel.
- A system provides **confidentiality**, if techniques for information restriction exist and are applied. The goal of the service is to provide data only for concrete entities while restricting the access of others. Access restriction can be realized on different levels of the OSI model: it is possible to restrict the access on the operation system level and introduce users with different privileges, or it is possible to create an Internet framework, where the access is restricted through a two-factor identification.
- **Authenticity** of the data states that the origin of data is known and verified. This security service ensures identification of the sender and postulates that no message can be sent by a foreign person using another identity. Sometimes the authenticity must not provide the identification of the sender, for example in the case that the sender wants to stay unknown. It is also important to note that authenticity not only creates security, but that security is also required in order to provide authentication. If a user knows a password, then he can use it and log-in into a service. In order to authenticate the user, the system must itself know either a password or the hash value of the password. If the system is compromised and the data is stolen from the server, it can be possible either to recreate the password, or recreate the hash value. The latter is possible in that case, when a non-secure hash algorithm (e.g. SHA1, SHA2 or MD5) is used on the server side [25], [26].
- **Non-repudiation** is the ability of the system to bind identities to actions and make this connection consistent in time. It must be impossible for an entity to perform an action and say afterwards that somebody else was the entity who did it. Non-repudiation is not straight-forward in a symmetric cryptography: if Alice and Bob share a secret key, both Alice and Bob could be authors of a secret message.

A channel cannot be called secure before providing at least one of these security services. When speaking about a secure channel, it is often useful to have a visualization of how the security services are mapped into the channel. Let us have a look at Fig. 1 and take the first formalization approach.

The figure shows Alice (A) communicating with Bob (B) and Charlie (C). All of them are surrounded by $evil$, hence Eve ($e_1, \dots, e_4$) may deploy several computers in order to catch the communication between Alice, Bob and Charlie. Bob and Charlie are on the mailing list of Alice, and Alice wants to send an important update regarding the stock prices on the two. The entities A, e_1, e_2, e_2 are potential senders of the message (we call them $\mathcal{S}$), and entities B, C, e_4 are potential receivers of the message (we call them $\mathcal{R}$). Let us use the function $a(m)$ in order to identify the author of a message m, the predicate $g(x, m, y)$ in order to say that x got a message m from y, and the predicate $p(x, m)$ in order to say that x has a permission to read the message m. Let m be a fixed message.

Now, Alice sends the stock update to Bob and Charlie. Alice establishes a connection to the both, hence "she creates a channel". If it is known and verified that Alice is going to send a message, then the channel provides authenticity. In other words,

$$\exists x \in \mathcal{R}.\ \exists! y \in \mathcal{S}. : a(m) = y \Rightarrow g(x, m, y). \qquad (1)$$

Otherwise, e_1 or anybody else could send this message. If it is ensured that only Bob and Charlie are going to get the message of Alice, then the channel provides confidentiality. In other words,

$$\exists y \in \mathcal{S}.\ \exists P \subseteq \mathcal{R}.\ \forall x \in P : g(x, m, y) \Rightarrow p(x, m). \qquad (2)$$

Otherwise, e_4 could also get this message and use it in order to manipulate the market. If it is known and verified that the data was not modified during the transport, then the channel provides data integrity. In other words,

$$\exists y \in \mathcal{S}.\ \exists x \in \mathcal{R} : g(x, m', y) \wedge a(m) = y \Rightarrow m = m'. \qquad (3)$$

Otherwise, e_4 could catch the message, modify it and send in the name of Alice. Finally, if Bob and Charlie can uniquely identify Alice as a sender, then the non-repudiation is ensured. In other words:

$$\exists x \in \mathcal{R}.\ \exists y \in \mathcal{S}. : g(x, m, y) \Rightarrow: a(m) = y. \qquad (4)$$

In fact, we have just established a basic logical framework for security services. However, implications (1) - (4) are not perfect. For example, (3) assumes that there is only one message in the network. In order to create a logical calculus for security services only, we have to refine all the four statements and introduce some new ones. It is worth mentioning that we have not done any cryptography yet. This example shows that even a formalization of four basic terms may become a challenge. But, let us go further.

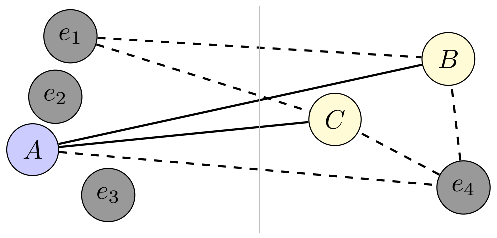

Fig. 1. Secure Channel

In order to accomplish these four security services, computer systems use a combination of non-cryptographic and cryptographic techniques.

For example, Cyclic Redundancy Checks [7] are aimed to provide data integrity, when sending data over a noisy channel, and are an example of a non-cryptographic technique. The technique is used in Ethernet on layer two of the OSI model. While catching errors resulted from a noisy channel, this technique cannot provide confidentiality or authentication, as data traffic can be sniffed before arriving at the destination. As a consequence, cryptographic techniques are needed.

No technique alone can fulfill data integrity, confidentiality, authenticity and non-repudiation at the same time. As a consequence, a combination of techniques is needed. Among most common cryptographic primitives are symmetric cryptography, asymmetric cryptography, hashing and digital signatures.

B. Symmetric cryptography

In symmetric cryptography Alice and Bob share a secret key and use it to encrypt and decrypt messages. Alice and Bob have to find a way to exchange this key before starting to transmit secret messages. If Alice has several communication partners, then she has to create a new key for each communication partner. Hence, a key management is required. At some point in time, Alice may decide to create a new key for the same communication partner. So, key management is aimed not only to create new keys, but update the old ones.

Symmetric algorithms can be used to achieve confidentiality, authentication, integrity and limited non-repudiation [6]. AES [8] can serve as an example of a symmetric algorithm. AES is very fast and can be used for the exchange of a big amount of data.

C. Asymmetric cryptography

Asymmetric cryptography is based on a different approach. Alice and Bob create pairs of a public and a private key.They share their public keys, and everyone can see them. However, only Alice and Bob know their corresponding private keys. If Alice wants to send a message to Bob, she takes the public key of Bob, uses it in order to encrypt her message and sends the encrypted message to Bob. He can use his private key in order to decrypt the message.

Asymmetric algorithms demand high computation effort, hence they are not suitable for a big amount of data. Instead, assymetric algorithms are used for digital signatures. In combination with symmetric algorithms they can be used for key transport and key management. This allows to achieve authentication, integrity and non-repudiation. In this approach, Alice would first exchange a key with Bob using an asymmetric algorithm, and then use this key for a symmetric one. A well-known example of an asymmetric algorithm is RSA [28].

D. Hashing

The main purpose of hashing is to reduce a message of an arbitrary length to a message of a fixed length. The resulting message is called the **message digest** and can be seen as a fingerprint of the message. The function that performs the reduction is called a **compression function**.

Cryptographic hash functions ensure the **Avalanche effect** by construction. This means that changing a single bit in the input message will result in a change of every bit in the output message with a probability more than or equal to 50%. Cryptographic hash functions also fulfill the properties of a **one-way-function**. That is, it is easy to compute the value of the function, knowing the functions' parameters, but it is almost impossible to reconstruct the parameters, knowing only the function value.

Together, Avalanche effect and the one-way property ensure that it is almost impossible to create two different messages that result in the same message digest[1].

The main purpose of a message digest is to provide data integrity. If Alice wants to protect the integrity of the message, she can compute the message digest and send it to Bob together with the original message. Bob can calculate the digest on his own and compare it to the one sent by Alice. If both values are equals, Bob can be sure that the message was not changed during the transaction.

E. Digital signatures

Hash functions can be used for signing the messages in the asymmetric cryptography. Alice can calculate the message digest and sign it with her private key. Later, everyone is able to calculate the message digest alone and use the public key of Alice in order to verify the authenticity of the message. This way, digital signatures provide data integrity, as well. If the data was changed, a different digest is produced.

Digital signatures also provide non-repudiation. If Bob can decrypt a signature with the public key of Alice, the message must be originally encrypted by Alice.

A complete introduction about cryptographic techniques and their influence on security goals can be found in [6]. But now, we have all the knowledge we need to define a public key infrastructure.

IV. Public Key Infrastructure

If Alice and Bob communicate directly, they can manage the communication and create keys on their own. But, if Alice needs to communicate with hundreds of colleagues, then the process of key management becomes very complex. Alice has to note down which key pair corresponds to which communication partner. Sometimes, different public keys may refer to the same identity, but be dedicated for different actions. For example, Alice can share her thoughts with Bob using one pair of keys, and send stock updates using the other pair. In this case, a number of keys used only by one entity can explode in size. Hence, a problem of key management becomes a challenge.

One of the possible solutions is to use a **Trusted-Third-Party** (TTP), an entity which will take care of the key management and allow Alice and Bob focus on the communication

[1]However, tools like "fastcoll" or "unicoll" are proven to be effective in length extension attacks on SHA2 or MD5.

itself. By providing key management services, a TTP binds public keys to identities. In a correct approach, a TTP can ensure all four security services mentioned earlier in this paper.

When a great amount of users is involved, even a single TTP can become overwhelmed. The problem is solved by enlarging the amount of involved TTP and binding them into an infrastructure. When this infrastructure is designed to distribute public keys, it is called a **public key infrastructure** (PKI).

PKI consists of several components, so that each component has a desired role in the whole infrastructure. Among basic PKI components are **certification authority** (CA), **registration authority** (RA), a repository and an archive.

A. *Certification authority*

The main purpose of a CA is to certificate keys and publish these certificates. To fulfill this, a CA verifies that the certificated identity holds a corresponding private key. The certificated identity can be either an end-user, or an another certification authority. If two CA certify each other, we speak about a **cross-certification**. Cross-certification is useful for establishing a network of trust, or when combining different certification networks.

The certificates of a CA usually include more information about the owner of the public key. This can include a contact address of the identity, expiration date of the certificate, intended use, the own digital signature of the CA and other information. All this data is formed into a data structure. The common data structure used nowadays is $X.509$.

If Alice wants to verify the certificate issued by a given CA, she can apply the public key of the CA to the certificate' signature. Usually, this verification is done by a software installed on the user's computer, so that no manual action is needed.

Not all certificates expire by their expiration date. In special cases, a private key of the identity may become public in the sense that the key gets known to further identities. In this case we say that the key was compromised.

For such cases, a CA maintains **certificate revocation lists** (CRL), which include the certificates that were revoked before their expiration date. As an example, a worker could leave his company, so that his key pair could no longer be in use. If a worker is still doing his job, but his private key was obtained by hackers, they could perform actions in his name. In this case, actions made by this worker can not be considered authentical anymore, and the certificate must be revoked. If a CA experienced a hacker attack, then the certificates issued by this CA can no longer be considered trustworthy, as well. Hence, a significant amount of protection is needed in order for PKI to stay trustworthy.

Certifications authorities sign their CRL-s, so that information contained there is considered valid and is protected against any manipulation, as well.

In a technical sense, a CA can be considered as a collection of hardware, software and the people who operate it [6].

B. *Registration authority*

Registration authorities take an intermediate role between an identity and the CA in the process of issuing certificates. A RA can be seen as an interface for the client. In order to get a certificate, entities make an application by the registration authority. The RA performs all necessary checks and validates the given information before forwarding this information to the CA.

A CA has a number of RA-s it works with. In this sense, a CA relies on the trustworthiness of the partnered RA-s. Being part of an own PKI, a RA also has a pair of public and private keys. Due their significant role in the infrastructure, the private keys of each RA must be stored securely, as well. This shows that the common security of the whole PKI is dependent not only on CA alone.

In a technical sense, a RA is organized the same way as a CA, being a collection of hardware, software and people who operate it.

C. *Repository*

A PKI-repository is aimed to store and manage certificates. A repository is implemented as a (possibly distributed) database and must be interactive. If Alice wants to retrieve an information about a certain certificate, a request to the repository is done. For the case that a CA wants to keep some certificates in secret, access restriction policies are needed.

D. *Archive*

An archive is used to store the certificates whose validity time has expired. Although it seems not necessary, an archive can be used at a later point of time in order to check the validity of a certificate that was issued before the expiration date of some other certificate. This way, the validity of a certificate to a certain point of time can be checked.

V. PKI ARCHITECTURES

We are going to make use of the graph theory in order to model the architectures of a public key infrastructure. In this context, we assign nodes to identities and express "trust" by directed edges.

A. *Hierarchical architecture*

Hierarchical architectures may be the most popular way of organizing a public key management. CA of a higher order issues certificates to low-order identities. The first CA in a hierarchy is called a **root-CA**. The whole structure forms a tree in a graph-theoretical sense. An example of a hierarchical infrastructure is shown in Fig. 2.

Every entity in the network knows a public key of a root-CA. If Bob wants to validate that the certificate of Alice is valid, he verifies the certificate of Alice, issued by D, then the certificate of D, issued by B, then the certificate of B, issued by A, which public key she knows. This **chain of trust** ensures the validity of the certificate.

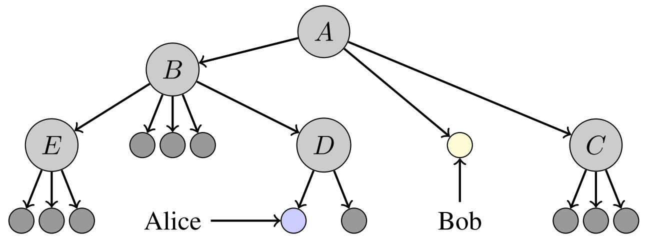

Fig. 2. Hierarchical Architecture

B. Mesh architecture

Mesh architecture considered a different approach. Here, CA cross-certificate each other. This results in a mesh architecture shown in Fig. 3. Different steps are required, if Bob wants to verify the validity of the Alice' certificate. Bob is holding a public key of C. He checks the certificate of Alice issued by E, then the certificate of E, issued by D, then the certificate of D, issued by C, which public key he knows. The validation is performed differently due to the cross-certification. One can easily see that the former hierarchical architecture is a special case of this architecture. In general, in order to transform a mesh architecture into a hierarchical architecture, one has to define a root and remove cycles by performing a minimum spanning tree search.

C. Bridge architecture

Bridge architecture is necessary, when different PKI-s should be linked together. In order to do this, a **link-CA** is introduced. This CA does not issue certificates to individual users, but is aimed to make a **trusted point** between different architectures.

This trusted point is going to establish trust relationships with other PKI-s depending on their type. The link-CA is going to make a link to a root-CA, if the PKI's architecture is implemented hierarchically, and is going to choose one of the CA-s, if the architecture is implemented in a mesh style. An example can be seen in Fig 4. Here, CA B, D and E form a mesh architecture, whereas F and C form a hierarchical one. Alice and Bob are in different architectures, but want to establish a connection. To do so, they have to verify their certificates. Alice trusts B, because she knows the public key of B and B is a root-CA in Alice' network. She trusts A, because B issued a certificate for A. She trusts F, because A issued a certificate to F, and she trusts Bob, because there is

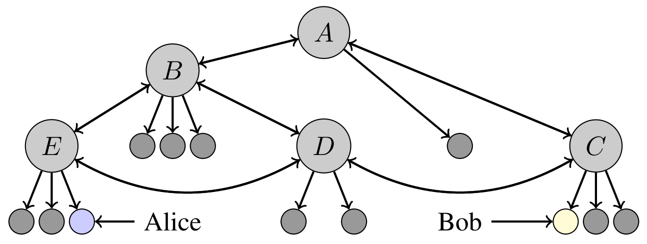

Fig. 3. Mesh Architecture

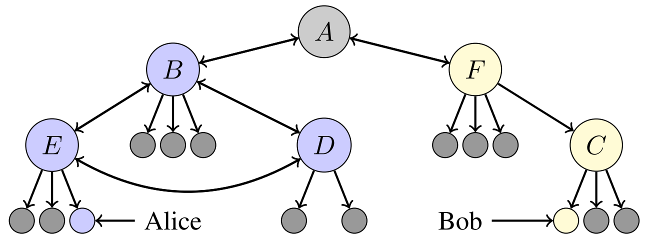

Fig. 4. Bridge Architecture

a valid certification path for Bob's certificate in Bob's PKI. Bob can verify the certificate of Alice in the same manner.

D. Web of Trust

Web of Trust (WoT) can be seen as a generalization of a "common" hierarchical PKI. While "mesh architecture" usually describes a form of a PKI architecture, where both CA-s and end-users can be certified by a former CA, the term "Web of Trust" describes an architecture where entities certificate entities. In other words, every entity can take the role of a CA. Every entity in the network can issue certificates.

This creates different views on the network, so that certification paths may differ. Alice can consider Charlie trustworthy and issue a certificate for his public key. However, Bob can either have different standards of the required trust level, or have bad experience with Charlie. Hence, he will never trust Charlie, but Alice will.

VI. Modeling trust relationships

Let us take the next formalization approach. For this, we are going to shortly discuss the extensive model on trust computation described in [3]. An interesting fact about this model is that it formalizes different levels of trust, which are then used to compute the trust value.

So, a trust of level 1 means that an entity is considered to be trustworthy in issuing certificates. A trust level of 2 means that an entity is considered to be trustworthy in performing recommendations of level 1, hence to be trustworthy in point on entities that are considered trustworthy in issuing certificates. Generally, the trust level $i + 1$ is defined by the trust of doing recommendations of the level i. In a certain sense, a certificate can be seen as trust of level 0.

In this model entities are represented by nodes. Edges represents different types of relationships, which can be of four different kinds:

- $Aut_{A,X}$: Alice believes that a public key of X is authentic, hence belongs to the entity X.
- $Trust_{A,X,1}$: Alice believes that X is trustworthy of issuing certificates.
- $Cert_{X,Y}$: Alice believes to hold a certificate for the public key of Y issued and signed by the entity X.
- $Rec_{X,Y,i}$: Alice holds a recommendation of level i for entity Y issued and signed by entity X.

Alice' **view** is formed by statements of these four kinds. Later, operations on these statements are introduced, allowing to

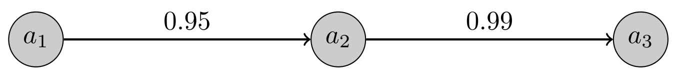

Fig. 5. Single Path

build a Kleene closure of the trust relationships from the Alice point of view.

The next important step of [3] is to assign numerical values in the range from 0 and 1 to the relationships contained in the Alice' view. Interpreting these values as probabilities in a well-defined random experiment allows us to use the whole specter of theoretical tools in order to make statements about the trustworthiness of the network entities.

The model presented in [3], however, has a significant drawback. Although it provides an expressive power of formalization, it is not suitable for direct applications. This drawback is pointed out by the authors. Hence, we are looking for a simplified approach that can allow us to compute trust in a real-world problem.

Such simplification can be found in [2], which is based on [3]. The given model assumes an existence of a PKI, where a global database holds public keys of participants. Participants put their signatures on identities that they believe are trustworthy. This results in a web of trust, where each participant may share publicly, which keys he believes to be authentic. This model is interesting, because a decentralized architecture is implemented by means of a central database.

The initial steps in [2] are similar to the previous model: the initial point of view of Alice is calculated and the probability values are assigned to the edges. Concrete examples are described below.

A. Single Path

Let us look at a very simple case in Fig. 5. Here, a_1 believes that the certificate of a_2 is authentic to 95%. At the same time, a_1 trusts the recommendations of a_2 to 99%. Hence, the resulting trust of a_1 in a_3 is just a product of probabilities: $0.95 \cdot 0.99 \approx 0.94$. Assuming that $T(a, b)$ means the trust of a in b and $L(c, d)$ stands for the probability of a link between c and d to be authentic, we can write the formula generally:

$$T(a_1, a_n) = \prod_{i=1}^{n-1} L(a_i, a_{i+1}), \quad n \geq 2.$$

B. Parallel Paths

A slightly different example is represented in the Fig. 6. Here, two different ways lead from a_1 to a_3, which intuitively means that the trust of a_1 in a_3 should be higher than in the single path case.

Assuming independence of the events, we have

$$P(A \cap B) = P(A) \cdot P(B)$$

for the events A and B. Then $(1 - 0.99 \cdot 0.95)$ calculates the distrust in the higher connection path, $(1 - 0.80 \cdot 0.95)$ calculates the distrust in the lower connection path. Their

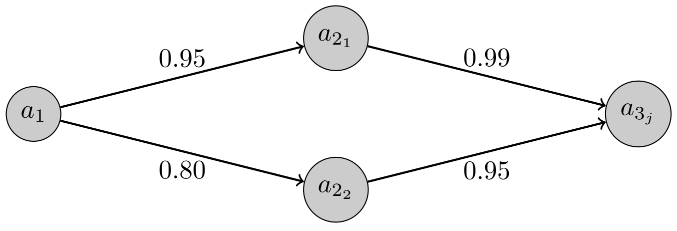

Fig. 6. Parallel Paths

product is the distrust in both connections. Hence a_1 thinks the public key of a_3 to be authentic with the probability

$$1 - (1 - 0.99 \cdot 0.95)(1 - 0.80 \cdot 0.95) \approx 98,6\%.$$

A formal notation is a bit more challenging. If there are m paths from a_1 to a_n, then let n_j be the length of the path j and let $T_j(a, b)$ be the trust in connection from a to b if taking the path j. Identify a_n with a_{n_j} for every j. The resulting trust is

$$1 - \prod_{j=1}^{m} \left(1 - T_j(a_1, a_{n_j})\right).$$

C. Interconnected Paths

At this point, calculations in papers vary and become interesting. Consider an example depicted in Fig. 7. Different interpretations of the given network are possible.

An optimistic approach would say that a_{2_1} reinforces a_1-s trust in a_{2_2}. In fact, that boosts the trust value of a_{2_2} and results in the statement that a_{2_2} must be a secure and trustworthy node.

However, a pessimistic approach could say that a_{2_1} uses a_{2_2} to reinforce a_1-s trust in a_{3_j} and to reinforce own trust in a_{3_j}. That can happen just because a_{3_j} may be a friend of a_{2_2} and wants to boost it's authority. The worst point about such a view is that a_{2_2} may not even know it. In this case, boosting a_{2_2} does not actually say anything about the trustworthiness of it, because a_{1_2} just wants to boost his trust in a_{3_j}. From an optimistic point of view, this is a catastrophe, because a_{2_2} gets a falsely boosted authority being instrumentalized for the authority of others. This example shows that game-theoretical approaches can be indeed interesting in modeling trust relationships.

In our example, the paths are no longer independent, hence the trust value cannot be calculated as easily, as in the previous examples. However, one possible solution would interpret

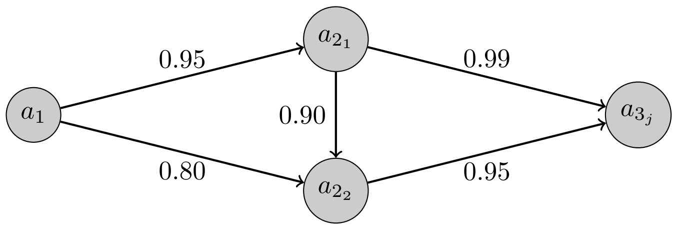

Fig. 7. Interconnected Paths

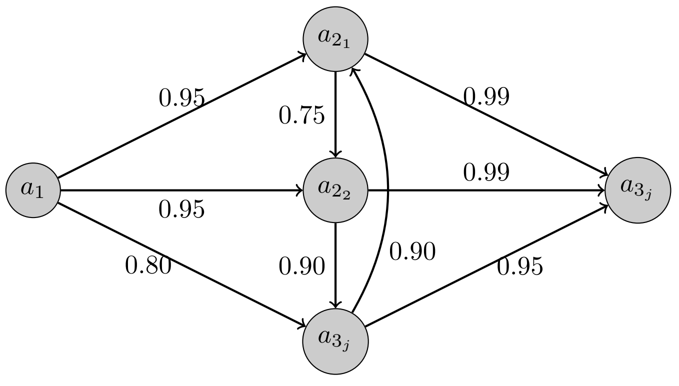

Fig. 8. Interconnected paths with loops

edges of the graph as components of a binary vector, "1" meaning an edge to be present, "0" zero meaning the edge to be absent. If a bit combination forms a valid path in the network, the probability of this path has to be calculated and accumulated in a common trust value while applying an appropriate combining technique. However, this approach will grow exponentially and is going to be calculated eternally starting by 100 entities.

D. Interconnected Paths With Loops

A complicated situation can be seen in Fig. 8. By introducing loops algorithmic calculations become no longer straightforward. However, [2] presents an interesting heuristics approach, that allows out-of-the-box calculations. Measurements show that the approach can be accurate if compared to the exact algorithm shown in [3] while being "fast" for a real approach.

Let us recall the questions we asked in the introduction. How do we choose, which nodes in the network are trustworthy? How do we calculate trust?

Imagine a real-world network depicted in Fig. 9. It is known that A wants to get some services from the network, while it is known that only E and F can fulfill these services (E and F could both be registration authorities while A wants to get a certificate, for example). Moreover, both E and F rely on B,I or H to fulfill their own services, and all B,I and H rely on the services of K,J or Z. How can we approach such a task?

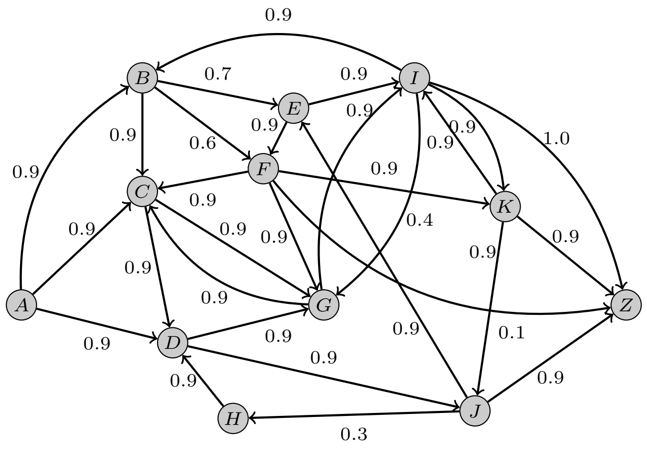

Fig. 9. Example network

First, we need to understand that A does not trust E and F directly. A may even not know them. As a consequence, the trust into E and F is formed by indirect trust relationships, and A relies on other entities in the network for estimating the trustworthiness of the both.

When the trust to the communication partners is known, we assume that A will choose the service with the best reputation, hence, with the maximal trust value. For simplicity reasons we will also assume that all the nodes share the same view - hence that link probability on the edges stay the same for every node in the network. That must not be the case: on the one hand, Fig 9 represents the view of A, the view of B could be different. On the other hand, view is task-dependant. If A wants to get a certificate, the view can take the form of the Fig. 9. But, if A wants to revoke a certificate, other nodes can get better probabilities, hence can be considered trustworthy for the task.

Secondly, A needs to compute the trust into E. We have implemented the Caronni algorithm [2] for this purpose. The implementation can be found in [29]. The core idea of the algorithm is to "divide and conquer". If A and some other node X are direct neighbors, the trust in X is a link between A and X. So, A trusts B to 90% in Fig. 9. If X is not a direct neighbor of A, we have to recursively compute the trust value into the neighbors of X and combine them probabilistically with the edge values leading to X. So, E has four direct neighbors: B, F, I, J. The edges between (E, F) and (E, I) are outgoing edges for E, hence computing the trust of A in F or I will not boost the trust of A in E. So, the trust of A in E is formed indirectly by the trust of B and J in E. In the same manner, the trust of A in J is formed by the trust of A in D and K. After executing the algorithm recursively for all nodes before getting to A and getting the trust of A in J, we can combine the value with the link (E, J) using the techniques from the previous section.

The recursive step simplifies the network. In order to compute the trust of A in both B and J (and later in both D and K), we temporarily remove E from the network, create two copies of this network and start the algorithm again, while setting B and J as new target nodes in the copies. This results in a multi-level recursion where we either compute a trust of A into a node X, or start as many recursive calls as there are ingoing edges for X. As we remove one node in each step, the algorithm terminates.

By applying the algorithm on the whole network in Fig. 9 we get the trust values $\approx 96.3\%$ for E and $\approx 94.4\%$ for F. As we assumed that A will choose the node with the highest trust value, A is going to choose E for fulfilling the service.

Finally, we can apply the same procedure for E and find out whether B,I or H is the node E trusts at most. Finding out that his trust in I lies by $\approx 99.7\%$ and is bigger than the trust in B or H, E will decide to choose the services of I. In

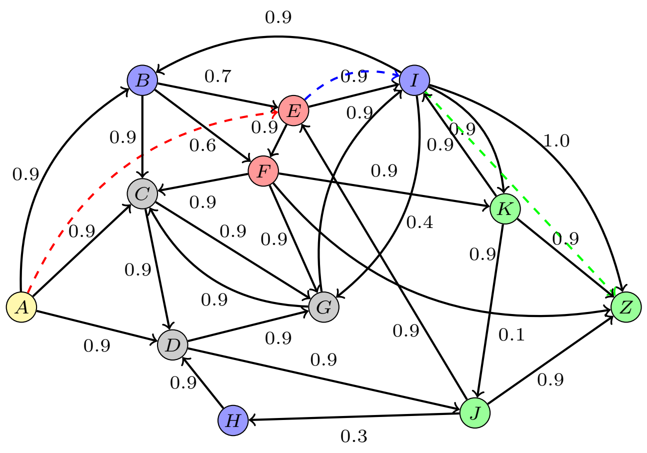

Fig. 10. Example network solution

the same manner, I is going to choose the services of Z. A complete solution path can be found in Fig. 10.

Let us recall the example described in the introduction. When applying for a credit, we can use Google to find a comparison portal. The browser is going to check the public certificate of Google, hence verify that we are dealing with Google, indeed. After entering the bank query, Google checks the certificates of the services and puts Check24 at the top of the results according to its own logic and view. A user can verify that the certificate for Check24 was issued by DigiCert by clicking the “lock” icon in the browser.

After clicking the Check24 link, we can enter the preferences and bounds for the credit. Now, Check24 has to find some banks for us. In this sense, Check24 relies on the services offered by the banks. It makes the search, checks the certificates of the banks and ensures that the certificates are valid and no fraud is possible. After choosing the bank, we can contact it for further information, while relying on Check24 that the bank exists and is verified. In this example, we relied on the services of our browser, which in turn relied on the services offered by Google, which in turn relied on services offered by Check24, which in turn relied on the services offered by the banks. We rely on the bank, because every intermediate node in the chain was considered trustworthy.

It is worth noting that we do not speak about the search rankings made by the search engines of Google and Check24. It is only important for us that all actors in our scenario went through some step of verification before appearing in the search results. That process binded their identity to their names and allows us to be sure that the services we use are indeed the services they claim to be.

VII. Evaluation of the work

We have tested the algorithm for different kinds of inputs, so that the problem of Fig. 9 could be solved in around $30ms$. However, the algorithm is not fast, if we consider the worst case. The worst case is a complete graph, where each node is connected with each other node. In this case, we would have $n \cdot (n-1)/2$ links in the network. In each recursion step we would drop one of the nodes, hence obtain a graph with $(n-1)\cdot(n-2)/2$ nodes. Continuing the recursion we can compute that

$$\frac{n\cdot(n-1)}{2}\cdot\frac{(n-1)\cdot(n-2)}{2}\cdot\frac{(n-2)\cdot(n-3)}{2}\cdots\frac{2\cdot1}{2}$$

$$= \frac{n\cdot(n-1)\cdot(n-2)\cdots2\cdot1\cdots(n-1)\cdot(n-2)\cdots2\cdot1}{2^{n-1}}$$

$$= O\left(\frac{n!\cdot(n-1)!}{2^{n-1}}\right).$$

Hence, the algorithm will need an unreasonable amount of time in order to compute the trust values for a complete graph and for a large amount of nodes. However, as all nodes in the network hardly trust every other node in the networks, this heuristics provides an efficient approach, - as long as the number of trust relationships stays bounded.

Although we have introduced a formalization of trust and have a working algorithm for trust computation, some of the disadvantages of these trust models must be mentioned.

A. *Certification authorities*

CA are known to be a single point of failure in PKI. If a CA is compromised, then the whole PKI infrastructure is in danger, as the certificates issued by the participants can no longer be considered authentic. We have also discussed that CA-s rely on RA-s while issuing certificates.

A prominent example of a CA’s failure is DigiNotar, which hosted a number of CA-s as a Trusted Third Party. DigiNotar issued certificates for SSL, certificates that could be used as a legal equivalent of a handwritten signature, and certificates for the Dutch government. DigiNotar suffered a breach during the summer of 2011, so manipulated certificates were issued. Among them, certificates for the `google.com` domain were found.

As a consequence, around 300.000 users were abused with Man-In-The-Middle (MITM) attacks. The traffic that was intended for Google subdomains was redirected to the attackers computers. The contents of the traffic became visible, so that Google credentials of the affected users could be stolen. An investigation has shown that the attack was initiated and directed from a certain territory. See the report in [4] for complete investigation.

B. *Transitive trust faults*

Another point of failure meets the way the certification path is built on its own. Imagine a chain of trust, consisting of authorities $T_1, \dots, T_9$, so that each authority T_i issues a certificate to an authority T_{i+1}, for $i = 1, \dots, 8$. Now, the trustworthiness of T_8 depends not only on the authority of a root, T_1, but also on the ability of $T_1, \dots, T_7$ to decide, which subsequent authorities can be considered as trustful. In this sense, T_1 cannot provide the same level of trust for different sub-certification levels.

A simple explanation would say that, if every level takes a certification decision with a 95% correctness and signs entity with a trust value being greater or equal than 90%, then the trustworthiness at the level 9 differs to $0.95 - 0.95^8 \approx 28,7\%$ if compared to level 1! Hence, the instance at level 9 would not be even directly certificated by the root-CA. This leads to a conclusion, that even though a certificate of an entity at the level x can be validated up to the root-CA, the root-CA cannot ensure the same level of trust as when it was issuing a certificate itself.

This corresponds to the fact that CA rely on RA before issuing a certificate. If one of the RA makes an error and considers a non-trustworthy instance as trustworthy, the whole infrastructure can collapse.

C. Certificate distribution to the end-user

Many users may consider the PKI architecture as a complex matter, or may not be able to verify the certificates on their own. For this case, a third party can make this work instead of them. It performs certificate validations, and this process stays hidden for the user. In fact, if we look back to our example in the introduction, certificates were checked many times after the user sent a query to `https://google.com` in his browser.

Let us assume that the user used Firefox browser for the query. At the moment, the browser comes with 155 pre-installed root certificates [9]. So, an end-user is totally dependent on Mozilla Corporation in determining which certificates should be considered trustworthy, while the Mozilla Corporation is dependent on the **CA/Browser Forum** [30], which introduces guidelines for establishing a global PKI. This adds an additional component in the chain of trust. Moreover, a certificate revocation can happen only with the corresponding browser updates. In this case, a danger persists, that the user is going to use the unsecure certificate even after the moment when the certificate was considered as compromised.

D. Log-based PKI

A common approach is described in [10]. **Log-based PKI**-s publish issued certificates on a log server. On the one hand, certificates are not considered valid, before they are published to the log server. On the other hand, by these means users can track the certificates on their own and reveal compromised certificates quickly. This technique corresponds to the idea of the open source community: transparency increases security, as many users are able to detect bugs or an unexpected behavior, which might stay unexplored if searched by a small number of people.

An example of a log-based PKI is the "Certificate Transparency" by Google [11].

However, a log-based PKI can also have drawbacks.

The system has to be centralized in order to provide consistent information. This is necessary in order to determine if a certificate was authorized for a particular domain [11].

A problem can also occur, if a CA is malicious itself. As a CA is the only instance which can revoke the certificates from the log, a malicious CA can potentially never do it. Responding to this threat needs time and costs effort. The work in [11] provides a decentralized PKI approach, which tries to resolve this issue.

E. Web of Trust

Recommendations is a key concept in WoT, as "it is impossible to know each communication partner in person" [3]. Hence, when acting in a decentralized PKI architecture, Alice will not only rely on her personal view on the network, but also delegate the trust propagation to potentially unknown nodes. PGP [12] is an example of the WoT approach.

Some authors [13], [14] claim that WoT has no way to deal with key revocation.

F. Blockchain

A blockchain-based PKI is another interesting alternative. The work in [13] prescribes to use blockchain for the "applications that require high reliability and full elimination of data manipulation risks".

Blockchain deals with the WoT problem of certificate revocation. As the information is distributed among peers, blockchain eliminates the single point of failure in the form of a CA, as well.

An interesting aspect is that it is impossible to delete information from blockchains. A positive consequence is that any misbehavior of a CA is going to be traced and well-documented. A negative consequence is that blockchain applications can potentially explode in size.

VIII. Conclusion

We have discussed the common ways of establishing trust in computer networks. Through the paper, we tried to show that different levels of formalization are necessary for that. In particular, every formalization technique is a subject for a separate work.

When establishing trust relationships, it is necessary to put the focus either on the computation effort, or on the theoretical power of the model. The introduced algorithm can be seen as a compromise for the both: although it can be theoretically slow and introduce some deviations from the exact mathematical calculations, it performs well in our examples and allows us to make statements about the trust relationships in the network.

The model can be extended in several ways. First, it is possible to assign dynamic trust values for the nodes. Trust values can either change by challenge/response actions, or age in time. In fact, the certificate validity date already implements this idea, so that the certificate issued at some time t_0 gets an aging factor of 1.0 after the expiration date t. However, it is possible to reduce both the expiration time and the factor by some value x. Then, the trust value at the time point $(t - t_0)/x$ will reduce by $1/x$.

On the other hand, it is possible to implement a dynamic view on the network, so that A can trust B to 50% for a task X, but trust B to 90% for a task Y at the same time. In fact, "different levels of trust" discussed in other works address this idea.

We hope that our work has shown both theoretical and practical approaches for trust calculations and want to thank our advisors for assistance.

References

[1] Li Yi, Weidong Fang, Wuxiong Zhang, Weiwei Gao, Baoqing Li. Game-Based Trust in Complex Networks: Past, Present and Future. Complexity, vol. 2021, Article ID 6614941, 2021.

[2] G. Caronni. Walking the web of trust. In Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000), pages 153–158, 2000.

[3] Ueli Maurer. Modelling a public-key infrastructure. In E. Bertino, editor, European Symposium on Research in Computer Security — ESORICS '96, volume 1146 of Lecture Notes in Computer Science, pages 325–350. Springer-Verlag, 9 1996.

[4] Hoogstraaten, Hans. (2012). Black Tulip Report of the investigation into the DigiNotar Certificate Authority breach. 10.13140/2.1.2456.7364.

[5] Ueli Maurer and Pierre Schmid. A calculus for secure channel establishment in open networks. In Dieter Gollmann, editor, European Symposium on Research in Computer Security — ESORICS '94, volume 875 of Lecture Notes in Computer Science, pages 175–192. Springer-Verlag, 11 1994.

[6] Kuhn, D.R., Hu, V.C., Polk, W.T., & Chang, S.H. (2001). SP 800-32. Introduction to Public Key Technology and the Federal PKI Infrastructure.

[7] Sheinwald, D., Satran, J., Thaler, P., and V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations", RFC 3385, DOI 10.17487/RFC3385, September 2002.

[8] Blumenthal, U., Maino, F., and K. McCloghrie, "The Advanced Encryption Standard (AES) Cipher Algorithm in the SNMP User-based Security Model", RFC 3826, DOI 10.17487/RFC3826, June 2004.

[9] https://ccadb-public.secure.force.com/mozilla/IncludedCACertificateReport

[10] S. Matsumoto, R. Reischuk. IKP: Turning a PKI Around with Blockchains. 2016.

[11] Laurie, B., Langley, A., Kasper, E.: Certificate transparency. RFC 6962 (June 2013)

[12] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007

[13] A. Yakubov et. al. A Blockchain-Based PKI Management Framework. In The First IEEE/IFIP International Workshop on Managing and Managed by Blockchain (Man2Block) colocated with IEEE/IFIP NOMS 2018, Tapei, Tawain 23-27 April 2018, 2018.

[14] J. Yu and M. Ryan, "Evaluating web pkis," in Software Architecture for Big Data and the Cloud. Elsevier, 2017, pp. 105–126.

[15] Ueli Maurer and Pierre Schmid. A calculus for secure channel establishment in open networks. In Dieter Gollmann, editor, European Symposium on Research in Computer Security — ESORICS '94, volume 875 of Lecture Notes in Computer Science, pages 175–192. Springer-Verlag, 11 1994.

[16] https://manpages.debian.org/bullseye/traceroute/traceroute.1.en.html

[17] Hornig, C., "A Standard for the Transmission of IP Datagrams over Ethernet Networks", STD 41, RFC 894, DOI 10.17487/RFC0894, April 1984

[18] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981

[19] Hunt, Ray. (2001). Technological infrastructure for PKI and digital certification. Computer Communications. 24. 1460-1471. 10.1016/S0140-3664(01)00293-6.

[20] Mashayekhy, L., Nejad, M.M., & Grosu, D. (2021). A Trust-Aware Mechanism for Cloud Federation Formation. IEEE Transactions on Cloud Computing, 9, 1278-1292.

[21] Mr. Jeelani; Kishan Pal Singh; Aasim Zafar. "A Trust Calculation Algorithm for Communicating Nodes in Wireless Sensor Networks". International Research Journal on Advanced Science Hub, 3, Special Issue ICARD-2021 3S, 2021, 145-152. doi: 10.47392/irjash.2021.083

[22] D. Lauterbach, H. Truong, T. Shah and L. Adamic, "Surfing a Web of Trust: Reputation and Reciprocity on CouchSurfing.com," 2009 International Conference on Computational Science and Engineering, 2009, pp. 346-353, doi: 10.1109/CSE.2009.345.

[23] Ritu and S. Jain, "A trust model in cloud computing based on fuzzy logic," 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2016, pp. 47-52, doi: 10.1109/RTEICT.2016.7807780.

[24] S. A. Soleymani et al., "A Secure Trust Model Based on Fuzzy Logic in Vehicular Ad Hoc Networks With Fog Computing," in IEEE Access, vol. 5, pp. 15619-15629, 2017, doi: 10.1109/ACCESS.2017.2733225.

[25] Stevens, Marc. (2006). Fast Collision Attack on MD5.. IACR Cryptology ePrint Archive. 2006. 104.

[26] Z. Al-Odat and S. Khan, "Constructions and Attacks on Hash Functions," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), 2019, pp. 139-144, doi: 10.1109/CSCI49370.2019.00030.

[27] https://en.wikipedia.org/wiki/Alice_and_Bob

[28] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016,

[29] https://github.com/bits4beethoven/wot

[30] https://cabforum.org/