End-To-End Encrypted IoT Devices

Snyder, Richard
rwsnyde2@ncsu.edu

Abstract

The Internet of Things (IoT) is becoming a pervasive part of in-home networks and systems. As those devices become more ingrained in users' lives, the requirements for the privacy of that data must become more robust. Sensitive user information such as live home video streams and personal health metrics should only be accessible by the parties they are relevant to. This paper attempts to create an authentication protocol and authority delegation scheme for IoT devices that is independent of primitive methodologies such as user accounts and passwords. Instead, this paper attempts to make use of a QR Code based Public Key Infrastructures (PKI) that implements mutual authentication between the receiving platform and the IoT device. This has resulted in a lightweight implementation of an end-to-end encrypted system that is independent of an untrustworthy Certificate Authority of cloud provider.

1 Introduction

In IoT, devices often use cloud platforms to run their processes, including authentication and bootstrapping. This reality is due to the limited amount of computational power that is available on-chip in those devices, as well as the number of integrated devices that a single user or set of users can own. But, in turning over control of those processes to cloud providers, sensitive user data can become compromised if the cloud provider becomes compromised. Because of those vulnerabilities, a lightweight authentication scheme that does not require the use of cloud-stored passwords must be implemented. In addition to authentication, devices should be accessible by multiple users of different authority levels. That requires an access control policy that is efficient in terms of disk space.

Previous works in this area have dealt with the limited computing capabilities of IoT devices. In a seminal paper by Zhao *et al* [29], a mutual authentication procedure using SHA-1 and feature extraction is showcased. The limitations to that work are that it relies upon an untrustworthy hash function in SHA-1. In the work presented by Doukas *et al* [11], an IoT-specific PKI is utilized to encrypt the data collected by devices and sent to a gateway. However, in that paper, the gateway sends the encrypted data to a cloud provider, which is not the intended scope of this research. We want our user's personal devices to be the only recipients of such data, not a cloud provider that will run big data analytics on that data.

In creating an appropriate delegation scheme, the work done by Anggorojati *et al* [2] is a great foundation that this research will rely heavily on. Despite its genius, their work deals with the use of federated devices, in that a hub of devices is managed by roles, which does not bode well for hubs that include devices that should be accessible and some that should not to a specific role.

In this paper, we introduce a mutual authentication protocol that utilizes QR codes as the public keys. QR codes are able to hold up to 3kb of data, which is in the same space complexity as modern RSA methodologies. The QR codes are lightweight enough that they do not require long processing times on the IoT device, yet they are robust enough that they can compare to long key-length hashes like SHA-3. In addition, our IoT hubs will be managed by certain *master* users that can bootstrap future user devices to their terminal IoT devices. Anggorojati's work introduced delegation requests from such *master* users, which we will build upon to bootstrap and assign roles to new user devices. Those delegation requests will contain signed data about the new device and the role that the new device should take on.

This paper makes the following contributions:

- A lightweight mutual authentication protocol for in-home IoT hubs utilizing QR codes
- A bootstrapping and access control scheme that is independent of Certificate Authorities

The remainder of this paper proceeds as follows. Section 2 provides an overview of the data structures and sequence of events that those processes incorporate. Section 3 describes the design of our paper. Section 4 evaluates our solution by viewing time-based metrics for those processes. Section 5 discusses additional topics and possible future work. Section 6 describes other related work. Section 7 concludes.

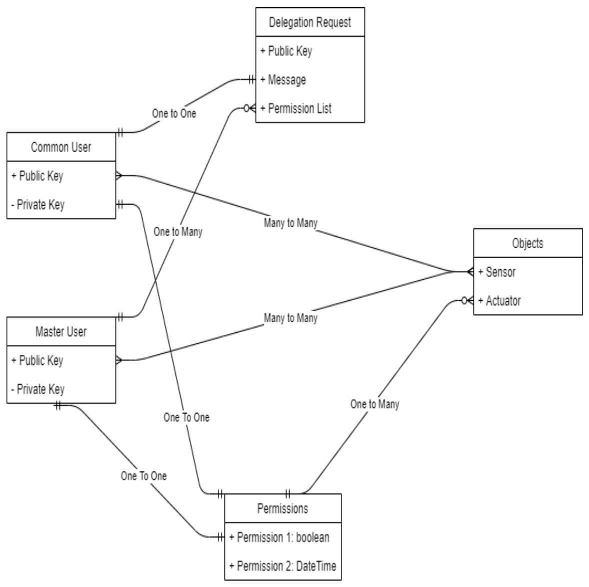

Figure 1: Relationships

2 Overview

The breadth of this research depends on the idea that *master* users are the ones that are conducting the maintenance of the accounts on the node. There are a few relationships that can be defined in this architecture, which will now be outlined.

First, there are the objects or individual pieces of data that we want to protect on the nodes. Those objects can be accessed by many individuals, and individuals can access many objects on a node.

Secondly, any *master* user can create a delegation request, but only one delegation request can be added per *common* user: the node will reject any public keys it already has in its table.

Finally, we must consider what permissions that an individual user has on the node to objects. These permissions will not be like file permissions in file systems, as the permissions on a node normally have to do with the functionality that node provides as a sensor or actuator. A user can have permissions to make a reading on a sensor and based on its reading, perform some action with the actuator, like turning the air conditioning on if the user sees the room is too hot. So, a user will have one set of permissions, and a node will hold sets of permissions equal to the cardinality of the user base.

A UML diagram depicting the relationships is given in Figure 1.

Now that we have a depiction of how the system is supposed to protect the node and its objects, we will describe the protocols that ensure they do so.

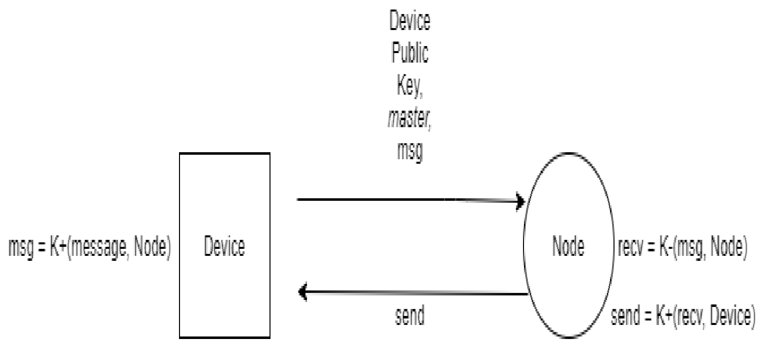

Figure 2: Bootstrapping the first Device to a terminal Node

3 Design

Our solution involves a mutual authentication scheme between the terminal IoT node and the end user device. Authentication begins when the first device attempts to establish a connection to the terminal node. The device will make an API call to the node that designates it wants to bootstrap, which will be different than the API call to add a new device to the node. If the node recognizes that it does not have any users in its table, it will thus designate that first user as the *master* profile. It will do so by storing the device's public key in a key-value-pair with a designation of *master* as the value. Considering the edge case where an initial *master* user already exists on the node, we can send back an unencrypted message to the device saying that it does not have the right to bootstrap a new device.

Because we must authenticate that the node that we are bootstrapping to is the correct device, the end of the bootstrapping process should require the node to send a response that proves the public key we used to access the node matches the IP address of the node we connected with. This can be done by encrypting a message using the node's public key, decrypting that message on the node, and then encrypting a response message using the device's public key. A pictorial description of the bootstrapping phase is shown in Figure 2.

Now that we have a single *master* user assigned to the terminal node, we can create new user profiles on the node by creating delegation requests. Because the public key of a new device is not yet known to the node, we must include that information in the delegation request. This will again help with mutual authentication in determining which node we connect to when creating a new profile. The process of adding a new user to the node is primarily the same as bootstrapping, with the main difference being that the *master* user creating the request can create a designation for the device: *master* or *common*. If the new user is designated at *master*, then the privileges will be unrestricted. If the user is designated as *common*, then the request should also include a list of privileges that user is allowed to have. If a value for a specific privilege is not given in the list, the default value assigned. These default values can be context specific, like having access to a node during certain times of the day.

Once a user has been given privileges in the node, they are now able to issue commands to the node. These commands should all have corresponding privileges assigned to them, so the node's API must include a getPrivileges() call, using the device's public key as the entry point into the local permissions table.

The getPrivileges() call will access the permissions table, which, in this implementation, is a dictionary of dictionaries. The inner dictionary contains keys of permission keywords and values of context specific values for those permissions. Each of those dictionaries is a value assigned to a key of the device's public key in the outer dictionary. The value of the permission can then be indexed by the device's public key and command name.

If the permission is False or out of the current context, then the node can simply respond to the device that the request has been denied. However, if the permission is granted, we must then authenticate that the user we're communicating with is actually the device whose public key is stored locally. Thus, we can send them an encrypted response message using their public key with a nonce and have them validate that nonce. It is a very simple form of mutual authentication, but it ensures us that a connection has not been established with a device that is spoofing the true device's public key.

Now that we have a proper understanding of how the system should function, we can now move onto evaluating how well our implementation conducts itself on a Raspberry Pi.

4 Evaluation

In determining if this is a viable solution for small devices in an IoT in-home hub, we must consider if the authentication protocol is lightweight enough to be completed in a short amount of time. In order to do this, we've conducted tests on a Raspberry Pi 4 with 2GB of memory at 1.5GHz. The implementation uses the **PyCryptodome**, a Python cryptography library, which has implementations of algorithms like RSA and AES. In addition to **PyCryptodome**, we use the **threading** module from the standard library in order to handle multiple connections at once on the server.

To test the performance of the Pi, we begin by allowing it to run our server code, which listens for a socket TCP connection from another device. When a connection is requested to the server, a new thread is spawned on the node that will handle the request. The data that is passed to the thread from the request will contain at minimum 5 pieces of data:

- command
- session key
- nonce
- tag
- device's public key

The command is in an encrypted form from the device using the node's public key and AES in EAX mode. The node will then use its private key to decrypt the command and determine what it should conduct from the list of API calls:

- Bootstrap the first node
- Add a new user
- Get a user's permissions

Because the command is encrypted by a device, every time we are servicing a request we are running decryption. That requirement will give us a good indication as to how much workload the node can handle and if there are any bottlenecks.

It is worth mentioning that we will not intend to prevent race conditions between threads. There is the possibility that two or more Bootstrap calls can be made and compete for which device will be designated as *master* first. We dismiss this possibility as it is highly unlikely that a malicious user will make a request to bootstrap their device at the same time the new owner attempts to bootstrap.

The variation that we are able to conduct in this experiment is two-fold: how many requests we are making to the APIs and how large the cryptographic keys are. First, because we are dealing with in-home IoT devices, it should be sufficient to say that less than 100 people will require access to a device at all, let alone at any one time. So, in order to test the true power of the node, we will reach that threshold in increments and see if the device is able to scale total request times linearly while maintaining an average processing time per API call. Second, because QR codes are able to hold about 3Kb of data, it is sufficient to attempt to exceed that metric and use an RSA public key that is of size 4096 bits, which is the current high threshold for asymmetric cryptography.

Tables 1 and 2 provide figures for the average and total processing time for two of the API calls: Add User and Get User Permissions. We have varied the number of concurrent requests from 10 to 100 and the RSA key size from 1024 to 4096.

These figures only take into consideration the amount of time it takes the thread to handle the decryption and

Add Users to Device			
Number of users	RSA Key Length	Avg. Time (s)	Total Time (s)
10	1024	.001558	.01558
10	2048	.001613	.0129
10	4096	.001824	.0153
50	1024	.001618	.07926
50	2048	.001648	.08241
50	4096	.001701	.10152
100	1024	.001618	.16175
100	2048	.00182	.18195
100	4096	.00201	.21002

Table 1: How long does it take to add users to the node?

Get User Permissions			
Number of users	RSA Key Length	Avg. Time (s)	Total Time (s)
10	1024	.003398	.03398
10	2048	.003498	.03498
10	4096	.003561	.03601
50	1024	.003259	.16297
50	2048	.004122	.20611
50	4096	.004925	.24349
100	1024	.004155	.41548
100	2048	.00752	.752
100	4096	.01246	1.002

Table 2: How long does it take to request your permissions?

API call. We've mitigated it this way to discount variable network latency between the device and the node. Despite that, the tables provide results that show sufficient evidence to the idea that processing time should not be an issue in in-home IoT devices. The average time to make API calls remains consistent as the number of requests grows tenfold. For example, the average time to add 10 users is within a millionth of a second of adding 100 users concurrently, in addition to having a near linear scaling of total time at an RSA key size of 1024.

The caveat to these figures is that they are run on a four core processor that runs the Raspbian flavor of Linux. The thread handling environment in Raspbian is most likely considerably better than the operating system that is running on most small IoT devices. Most Linux distributions take advantage of thread context switches during idle time, so multiple threads are able to work on their individual data while another thread is waiting for system calls to return with interrupts. In addition, unless the device has a large chassis like a thermostat, it is unlikely that it will contain a multicore processor in favor of a unicore processor. Therefore, the number of threads that can be run concurrently will be truncated to one instead of four.

5 Discussion

As stated in the previous section, we did not account for the network latency in the timing of our protocol. That should be a topic of study in future work to describe the difference in access time between users that are requesting their data from in the home, and therefore on the same LAN as the node, and those users that are accessing their node from outside their home. In addition, we ran this implementation on a Raspberry Pi, which is, at least in the IoT sector of devices, quite powerful. It is powered by an AC outlet and has four cores to process information. Future work should be dedicated to compiling a suite of IoT devices and seeing if their performance decreases linearly with their processor speeds and number of cores.

One thing that was mitigated from this implementation was the persistent storage of data. All of the public keys and permissions are stored in Python dictionaries in memory. So, when the server instance shuts down, so too does all of the public key and permissions information, which would require an entire reboot of the system with a new bootstrapping and delegation phase. Mitigating this with encrypted files containing the necessary data for authentication should be implemented in future versions.

In addition, privilege change is not handled in this implementation. There is currently no mechanism by which a user can have their privileges added or revoked, which should be necessary if someone's requirements for the node change. But, when implementing privilege changes, there must also be mechanisms that keep unauthorized users from escalating their privileges or depricating another's privileges unnecessarily. The mechanisms to do so are a topic of systems security that is well versed, so implementing such a feature is possible.

Discussion (discuss some of the important simplifying assumptions, and suggest possibilities for future work)

6 Related Work

At the basis of this project are Internet of Things devices that are managed within an in-home network. Internet of Things devices have multitudes of security concerns, of which previous studies have identified and attempted to mitigate.

In the paper introduced by Mahmoud *et al* [20], the IoT architecture is described as having three layers that provide different functionality to the devices they are enabling. The *Perception Layer* is the layer in which data is acquired by the devices from the outside world. In the case of a smart home camera, it would be the physical lense that receives images and the CPU that processes them. Then, when that data is formatted and processed, it is sent to the *Network Layer*, where the data is relayed to IoT hubs and other devices. In a smart home camera, the network layer would be responsible for handling the transmission of a video stream from the camera to the viewing device of that stream. Finally, the *Application Layer* is where the CIA triad of security is guaranteed on the device.

Mahmoud *et al* then go on to explain that there is a necessity for a lightweight key management system that allows for authentication between possibly unassociated devices. This makes sense because some nodes in the network might request data from another device it has not spoken to, or a new mobile device would like to receive the data collected by that device for viewing or further processing. The necessity for lightweight implementations is due to IoT devices not having large amounts of usable memory for their security applications.

Thus, the seminal paper from Zhao *et al* [29] explains such a lightweight, mutual authentication scheme. They begin the paper by asserting that SHA-1 and MD5 are insecure because of their collision space, so there must be a new formulation for authentication. Despite SHA-1's flaws, the paper combines it with feature extraction to ensure those faults are avoided; feature extraction provides the one-way property, which nullifies the problems associated with SHA-1. The paper makes the assumption that public and private keys are agreed upon prior to the distribution of the terminal nodes by the device distributor, which is exactly the assumpition that this research will be taking. The paper also makes the assertion that there need only be security in the transmission of the platform's private key, which, again, is exactly what this research will attempt to incorporate.

There are a number of papers that successfully compose key exchange algorithms that are secure [10, 11, 16]; however, those papers often rely upon digital signatures for integrity, certificate authorities for authenticity, or even specialized hardware [4]. What this research wants to utilize, though, is the assumption that QR Codes can be safely used by manufacturing processes to encode public keys for the terminal devices. QR codes are shown to be trustworthy in creating encrypted containers of information [3, 22]. If it can be assumed that QR codes can accurately represent data, then we can assume that manufacturers can accurately print and manage public key infrastructures through those codes.

As part of the requirements for usability and accessibility in IoT devices, there must be some sort of hierarchical structure that dictates priviliged users and restricted access. There are already large bodies of research in access control across other fields of computation like banking [23] and peer-to-peer networks [14]. However, these delegation schemes do not work particularly well with IoT networks because of their resource constraints, as previously mentioned in the authentication schemes. As part of a paper by Anggorojati *et al* [29], a Capability-based Context Aware Access Control (CCAAC) model was created. There is a key distinction that should be made between this research and the research of Anggorojati, in that they believe in the web-based model of federated devices, such that in different domains, users should have the same access throughout. In this research, there will not be a distinction made between different domains, because there won't be cross-domain access. Say, for instance, that there is a camera and a thermostat in a person's smart home network. The users that are delegated as ***master*** users should have unrestricted access to the whole suite. In contrast, users that are delegated as ***restricted*** should be given specific restrictions per device, and thus require specific authentication per device.

Now considering how CCAAC works, there is the presumption that a trust relationship has been established between a ***delegator*** *S* and a ***delegatee*** *D*. That assumption will persist in this research, considering that anyone who is being given access to in-home devices should be a trusted party and owner of the receiving device. When attempting to create a delegation profile for *D*, *S* will communicate with the terminal device using a signature signed by its private key to authenticate the delegation request is legitimate. From that request, the available policy list that the terminal device contains would be indexed to provide the specific permissions to *D*. Upon successfully finding a policy for *D*, a success packet is sent back to *S*, in which case *S* send the public key for the device to *D*. *D* can then mutually authenticate with the terminal device, thus creating a trust relationship between them. That body of work does not consider implementing timeout periods for the authenticity of party *D*, which could be an additional feature implemented in this research.

The inspiration behind this research began with the Signal protocol. A formal proof of security was written for the protocol by Cohn-Gordon *et al* [7], which can also lay some of the groundwork for this research. Signal utilizes what are known as *ratchets*, which are session keys that are updated on every message passed between the authenticated users. Trust relationships between users begin by one party sending, asynchronously, a batch of ephemeral public keys. When the first sender in the exchange wants to communicate, they take one of those keys and combine it with a long-term key to create the first message encryption key. Those messages keys are then *chained* together in all subsequent messages between the parties. One major implementation detail of Signal that will not be utilized is the vast categorization of keys, of which Signal has 10. This research will stick to asymmetric cryptography standards based solely on public-private key pairs. However, there could be use of Signal's implementation of HMAC-SHA256 for key generation. HMAC-SHA256 is combined with AES256 to create an encrypt-then-MAC scheme. Another distinction that will be apparent between this research and Signal's is the requirement for key registration in key distribution servers. It is not clear at this time how exactly the key storage will be managed across devices, but decentralization is preferred in such a distributed system.

7 Conclusion

The results from this work show that processing power and memory requirements can be kept at a minimum while still providing cryptographic excellence in IoT devices. Cloud providers will still be the gateway for big data analytics in production environment, but for in-home IoT devices, there is no reason why a the cloud should be the proprietor of authentication between user devices and hub nodes.

This work is limited, though, to devices that are plugged into a constant power supply and not to devices that potentially run on battery power. The electrical consumption to perform those cryptographic instructions should be a topic of research in future works. Another topic of interest should be how to write mutual authentication details into the QR codes so that the entire public key isn't being used as the encryption methodology, but only a small segment of the key.

Overall, in-home IoT devices will begin to collect immense amounts of data on users, both those who are aware and unaware of the fact. That data should only be accessible to those who actually own the data being collected, and compromises should not be made on how we securely authenticate who is receiving the data.

References

[1] D. Altolini, V. Lakkundi, N. Bui, C. Tapparello, and M. Rossi. Low power link layer security for iot: Implementation and performance analysis. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 919–925, July 2013.

[2] B. Anggorojati, P. N. Mahalle, N. R. Prasad, and R. Prasad. Capability-based access control delegation model on the federated iot network. In *The 15th International Symposium on Wireless Personal Multimedia Communications*, pages 604–608, Sep. 2012.

[3] J. F. Barrera, A. Mira, and R. Torroba. Optical encryption and qr codes: Secure and noise-free information retrieval. *Opt. Express*, 21(5):5373–5378, Mar 2013.

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In B. Preneel and T. Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 124–142, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[5] E. Bertino and R. Sandhu. Database security - concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, 2(1):2–19, Jan 2005.

[6] T. Borgohain, U. Kumar, and S. Sanyal. Survey of security and privacy issues of internet of things. *International Journal of Advanced Networking and Applications*, 6(4):2372–2378, Jan 2015. Copyright - Copyright Eswar Publications Jan-Feb 2015; Document feature - Diagrams; ; Last updated - 2015-05-22.

[7] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 451–466, April 2017.

[8] J. Crampton and H. Khambhammettu. Delegation and satisfiability in workflow systems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, page 31–40, New York, NY, USA, 2008. Association for Computing Machinery.

[9] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202, May 2002.

[10] M. Di Raimondo, R. Gennaro, and H. Krawczyk. Deniable authentication and key exchange. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 400–409, New York, NY, USA, 2006. Association for Computing Machinery.

[11] C. Doukas, I. Maglogiannis, V. Koufi, F. Malamateniou, and G. Vassilacopoulos. Enabling data protection through pki encryption in iot m-health devices. In *2012 IEEE 12th International Conference on Bioinformatics Bioengineering (BIBE)*, pages 25–29, Nov 2012.

[12] M. A. Khan and K. Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395 – 411, 2018.

[13] N. Kobeissi, K. Bhargavan, and B. Blanchet. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 435–450, April 2017.

[14] S. R. Kruk, S. Grzonkowski, A. Gzella, T. Woroniecki, and H.-C. Choi. D-foaf: Distributed identity management with access rights delegation. In R. Mizoguchi, Z. Shi, and F. Giunchiglia, editors, *The Semantic Web – ASWC 2006*, pages 140–154, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[15] C. Kudla and K. G. Paterson. Modular security proofs for key agreement protocols. In B. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005*, pages 549–565, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[16] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In W. Susilo, J. K. Liu, and Y. Mu, editors, *Provable Security*, pages 1–16, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[17] F. Li and P. Xiong. Practical secure communication for integrating wireless sensor networks into the internet of things. *IEEE Sensors Journal*, 13(10):3677–3684, Oct 2013.

[18] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, Feb. 2003.

[19] Z. Li, X. Yin, Z. Geng, H. Zhang, P. Li, Y. Sun, H. Zhang, and L. Li. Research on pki-like protocol for the internet of things. In *2013 Fifth International Conference on Measuring Technology and Mechatronics Automation*, pages 915–918, Jan 2013.

[20] R. Mahmoud, T. Yousuf, F. Aloul, and I. Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341, Dec 2015.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *Computer*, 29(2):38–47, Feb 1996.

[22] K. Saranya, R. S. Reminaa, and S. Subhitsha. Modern applications of qr-code for security. In *2016 IEEE International Conference on Engineering and Technology (ICETECH)*, pages 173–177, March 2016.

[23] A. Schaad, J. Moffett, and J. Jacob. The role-based access control system of a european bank: A case study and discussion. In *Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies*, SACMAT '01, page 3–9, New York, NY, USA, 2001. Association for Computing Machinery.

[24] A. Singla and E. Bertino. Blockchain-based pki solutions for iot. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 9–15, Oct 2018.

[25] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani. Network-level security and privacy control for smart-home iot devices. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 163–167, Oct 2015.

[26] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, and C. Lung. Smart home: Integrating internet of things with web services and cloud computing. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 317–320, Dec 2013.

[27] W. Wen, L. Wang, and J. Pan. Unified security model of authenticated key exchange with specific adversarial capabilities. *IET Information Security*, 10(1):8–17, 2016.

[28] R. Zhang, H. Ma, and Y. Lu. Fine-grained access control system based on fully outsourced attribute-based encryption. *Journal of Systems and Software*, 125:344 – 353, 2017.

[29] G. Zhao, X. Si, J. Wang, X. Long, and T. Hu. A novel mutual authentication scheme for internet of things. In *Proceedings of 2011 International Conference on Modelling, Identification and Control*, pages 563–566, June 2011.

[30] K. Zhao and L. Ge. A survey on the internet of things security. In *2013 Ninth International Conference on Computational Intelligence and Security*, pages 663–667, Dec 2013.