Keeping Connected in Internet-Isolated Locations

Kirsten Lunde Skaug, Elise Breivik Smebye, Besmir Tola, and Yuming Jiang
Department of Information Security and Communication Technology
NTNU, Norwegian University of Science and Technology
Trondheim, Norway

***Abstract*—In many scenarios, Internet connectivity may not be available. In such situations, device-to-device (D2D) communication may be utilized to establish a peer-to-peer (P2P) network among mobile users in the vicinity. However, this raises a fundamental question as is how to ensure secure communication in such an infrastructure-less network. In this paper, we present an approach that enables connectivity between mobile devices in the vicinity and supports secure communication between users in Internet-isolated locations. Specifically, the proposed solution uses Wi-Fi Aware for establishing a P2P network and the mTLS (mutual Transport Layer Security) protocol to provide mutually authenticated and encrypted message transfer. Besides, a novel decentralized peer authentication (DPA) scheme compatible with Wi-Fi Aware and TLS is proposed, which enables peers to verify other peers to join the network. A proof-of-concept instant messaging application has been developed to test the proposed DPA scheme and to evaluate the performance of the proposed overall approach. Experimental results, which validate the proposed solution, are presented with findings and limitations discussed.**

***Index Terms*—Device-to-Device Communication, Peer-to-Peer Network, Wi-Fi Aware, mTLS, Mobile Social Network, Decentralized Peer Authentication.**

I. Introduction

Smartphones have become essential tools for everyday tasks and are widely used for a multitude of communication services. Urged also by social distancing due to the global pandemic, mobile social applications such as instant messaging or web-conferencing are becoming indispensable services in our day-to-day life, and the vast majority of them are also offered through mobile applications. However, in out-of-coverage areas such as the remote locations or in areas where the network infrastructure is damaged, mobile devices are unable to establish a connection to a server and services become unavailable. This can be critical during natural disasters and rescue operations where user's connectivity can make a significant difference [1], [2].

Modern smartphones are equipped with different radio interfaces that can enable wireless communication among devices in proximity even in the most remote out-of-coverage locations or in cases where cellular outages are experienced. Common supported technologies include Bluetooth and Wi-Fi but Wi-Fi outperforms Bluetooth-based solutions for both high bandwidth requirements and also range coverage. The latest version of Bluetooth, i.e. 5.3, offers a maximum of 2 Mbit/s [3], which is several orders of magnitude less than Wi-Fi. While most of the Wi-Fi standards require the presence of a centralized coordinator, e.g., Wi-Fi infrastructure mode or Direct, Wi-Fi Aware, also known as Neighbor Awareness Network (NAN), enables decentralized peer-to-peer (P2P) connectivity between devices [4]. In the remainder, we use NAN and Wi-Fi Aware interchangeably. It offers a power-efficient and quick way of forming independent P2P networks that enable data transfer in a publish-subscribe pattern over TCP/IP [4]. Native support for Wi-Fi Aware has been already introduced in the Android 8.0, and later, operating system. Today, more than 80% of Android devices support Wi-Fi Aware [5].

While Wi-Fi Aware can use IEEE 802.11 Wi-Fi Protected Access 2 (WPA2) frame encryption to provide data link layer security [4], additional security levels need to be built on upper layers to secure application layer communication among users, because Wi-Fi Aware specifications consider such security as application-specific and independent of IEEE 802.11 MAC security [6].

Recently, Almon *et al.* [7] identified various exploitable security vectors affecting Wi-Fi Aware networks. In particular, they observed that unauthorized peers can easily assume the master role, which in turn may lead to various kind of attacks such as man-in-the-middle and node synchronization. This emphasizes *the need to integrate strong authentication mechanisms in the upper layers for Wi-Fi Aware-enabled applications.* [8] builds upon mutual Transport Layer Security (mTLS) an authenticated and secure out-of-coverage instant messaging solution using Wi-Fi Direct. Pre-signed up users can exploited online authentication to establish offline P2P networks and set up secure and integrity-protected communication. However, the solution is unfit for authentication of new users in out-of-coverage situations, hence limiting the use only to authenticated users before Internet connectivity loss.

To this end, the objective of this paper is to design, implement and validate a solution for application layer security when there is no Internet connection. The proposed solution exploits Wi-Fi Aware, as the technology for offline communication, embeds mTLS to provide mutual authentication and encrypted data exchange among peers, and also integrates a decentralized authentication scheme, well aligned with the decentralized nature of Wi-Fi Aware, which enables peer authentication (PA) also for offline scenarios. In order to validate the solution, a proof-of-concept instant messaging application, publicly available [9], has been developed and experimentally evaluated. An investigation that compares the proposed solution with similar technologies such as Wi-Fi Direct is also performed.

The remainder of the paper is structured as follows: Sec-

tion II presents a brief overview of Wi-Fi aware. Section III illustrates the proposed system architecture for enabling secure and trustworthy communication over Wi-Fi Aware. The implementation on a real testbed of smart devices running the Android OS is presented in Section IV. Successively, the validation of the different security layers adopted in the architecture and the experimental results analysis, in terms of both computation and connection times, together with a comparison to other related technologies, are presented in Section V. Finally, Section VI concludes the paper.

II. Wi-Fi Aware Overview

Wi-Fi Aware-enabled devices can autonomously form P2P network clusters and advertise services to facilitate inter-peer communication [4]. A Wi-Fi Aware cluster comprises a group of devices listening on the same channel and synchronized to the same clock. A Discovery Window (DW) specifies the scheduling on which devices wake up to listen for or advertise services. Synchronization allows the devices to wake up simultaneously for a short interval for sending and receiving data, thereby reducing battery consumption [4]. First, devices discover nearby clusters by listening for discovery beacons transmitted outside the DW by a device operating in a master role (step 1 of Fig. 1(a)). If a cluster is not detected, the device establishes a new cluster and starts transmitting discovery beacons advertising its presence. If a cluster is detected, the device receives synchronization information about time frames and channels (step 2 of Fig. 1(a)). The device responsible for the synchronization of the DW is called the Anchor Master (AM) and it propagates discovery and synchronization beacons, which can be a resource demanding task. Thus, to achieve fairness, this role is periodically alternated among peers. The last step is the service discovery process, where devices can publish a service and/or subscribe to one that matches their needs.

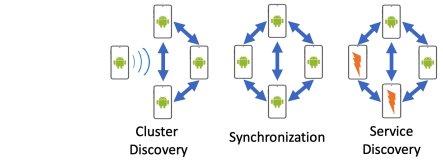
Before establishing a communication data path, the devices must agree upon which devices use its publish session and which use its subscribe session in order to avoid setting up two data paths. The subscriber initiates a 4-way handshake by requesting the data path set up. The data path is secured using IEEE 802.11 Wi-Fi Protected Access 2 (WPA2) frame encryption to protect the data and action frames being exchanged [4].

III. Architecture of the Solution

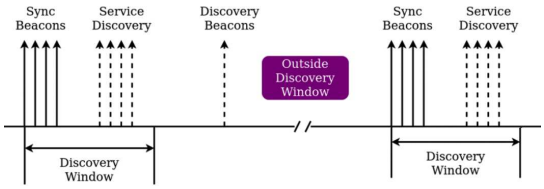
This section presents the system components for facilitating secure communication between users in a Wi-Fi Aware cluster.

A. The Authentication Server

The solution for the authentication server is in accordance with the Public Key Infrastructure (PKI), as used in TLS [10]. The authentication server acts as a Certificate Authority (CA). It is responsible for issuing, signing and revoking the digital certificates of new users of the application. The CA is a single Trusted Third Party (TTP) among the members of the social application. This authentication component is only reachable via the Internet and signs certificate signing requests (CSRs) according to the X.509 standard [11].



(a) Cluster discovery, synchronization and service discovery.



(b) Discovery window during which devices discover clusters, synchronize and discover services.

Fig. 1. Wi-Fi Aware Cluster formation.

1) Certificates: A certificate binds an identity to a public key. Each certificate contains the subject's public key, in addition to the certificate subject, the issuer and the expiry date. The public key is part of a public-private key pair generated when the application is downloaded. The certificate contains a signature signed by either the CA or the certificate owner (self-signed), depending on whether the user signed up for the application service before going offline.

2) Sign up Process: When a user connects to the authentication server and signs up to the application service, a Certificate Signing Request (CSR) is created using the key pair generated during download. The key pair consists of a private and a public key. The private key is only accessible to the user who is signing up. The CSR contains information such as country, email address, common name and public key.

If the authentication server considers the information provided in the CSR to be correct and trustworthy, the certificate request is approved, and a certificate is generated. The certificate is then signed by the CA and returned to the user who requested the certificate. In case users violate the terms and conditions for using the service or if a public key has been compromised, the certificate should also be revoked. To revoke credentials after the fact, a Certificate Revocation List (CRL) can be maintained [11].

B. Client-Server Components

All devices participating in a Wi-Fi Aware network constantly have a server running, awaiting incoming requests from clients. No device acts as an access point for forwarding messages for other devices. Thus each device must have a server running in order to be reachable. Consequently, each device must be able to work as both a client and a server. The device that first initiates a connection to another device is assigned the client role, i.e., the service subscriber.

1) The Application Server: The server is responsible for awaiting any incoming requests from clients at a specified port and must handle multiple connections to clients simultaneously. Each device runs a server on a unique port, making

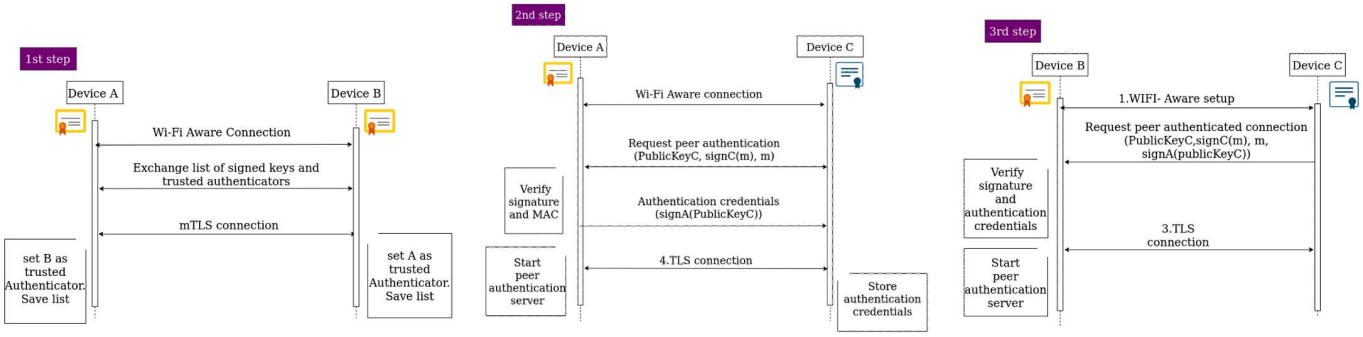


Fig. 2. Sequence diagram showing the steps necessary to set up a PA connection between two devices, B and C.

every device reachable for communication. Data exchange between a server and a client is over TCP/IP using the Wi-Fi Aware data path and the port advertised by the server. When a server receives an incoming request from a client, mutual authentication and the negotiation of cryptographic protocols are performed using mTLS before setting up a connection. When using mTLS, both server and client exchange certificates to verify that both parties have signed up for the service. The server will not accept communication sessions without the client first providing a certificate signed by the CA.

2) The Application Client: The client is responsible for requesting a session from a server and providing it with a certificate. A network can comprise multiple devices. Thus, a device must be capable of handling multiple server connections. The user initiating the communication session with another peer is automatically assigned the client role for that specific session. The port used to contact the server is exchanged during the establishment of the data path. The client secures its communication to the server by only accepting certificates signed by the CA. The cryptographic protocol specifies the encryption schemes supported by the client, which are provided to the server during connection setup.

C. Peer Authentication Components

This work proposes a decentralized peer authentication (DPA) scheme inspired by the technique proposed in [12], which enables users to be verified by another CA-authenticated peer. The technique is incorporated into the solution by using a server that does not rely on a certificate signed by the CA. The scheme provides unauthenticated users with authentication credentials, instead of a CA-signed certificate.

1) Peer Authentication Server: The peer authentication (PA) server uses authentication credentials provided by the client, instead of a certificate signed by the CA, to verify that another peer has authenticated the client. The credentials are provided to the application server before starting communication. If the client's credentials are valid, the PA server is started using a different port from the application server. The client is then able to communicate with the application server over a mutually authenticated and secure connection. If the client authentication credentials provided to the server are invalid, the PA server will not be started.

2) Peer Authentication Client: The peer-authenticated client is responsible for providing the PA server with authentication credentials, and requesting a PA connection. By providing authentication credentials, the client proves that another authenticated and trusted user has vouched for its identity. A device that has not signed up to the service before moving to an out-of-coverage area will become the client in any client-server connection. A peer-authenticated client operates in the same way as a regular client and a connection requires a PA server to provide a valid certificate. The client provides the server with its supported cryptographic protocols for negotiation.

3) Peer Authentication Credentials: Different from [12], our proposed solution uses only public keys to verify that an authenticated user has provided a signature. Public keys are shorter in size compared to certificates and can be sent using Wi-Fi Aware messages. Unlike the scheme in [12], we limit PA to one level, meaning that peer-authenticated users cannot authenticate other users. This is to ensure easy tractability of peer authentication and avoid abuse of it.

4) Peer Verification: A user without a CA certificate can request authentication from another peer in the network. Similarly, any device with a certificate signed by the CA can verify another peer. When a CA-authenticated user receives a request from a user who wants to be authenticated, it can deny or accept the request. Such a decision depends on the authenticator's trust in the validity of the identity presented by the peer requesting authentication. The definition of level of trust for each authenticator is disregarded in this work and is left for future work.

Fig. 2 illustrates how the proposed decentralized peer authentication (DPA) scheme works, specifically how an authenticated device, A, verifies an unauthenticated device, C, thereby enabling B and C to set up a PA TLS connection. The detailed idea and steps are as follows.

If a CA-unauthenticated peer (e.g., C) wants to communicate with an authenticated peer (e.g., B), it must first be vouched for by a CA-authenticated peer (e.g, A), shown as Step 2 in Fig. 2. If A accepts the request from C , it will release authentication credentials to C, making it a new authenticated peer. These credentials will enable C to prove that it has been vouched for by A in future interactions and thus is allowed

to participate in authenticated and secure communication, e.g. with B as shown in Step 3 in Fig. 2. If the request is denied, C cannot participate in any connections for communication, but it may try to request authentication from another peer in the network.

For user B and the peer-authenticated user C to communicate (refer to Fig. 2), B must verify that user A is a CA-authenticated user who is trusted to verify C. Thus, A, the authenticator, and B must have had a secure and mutually authenticated connection beforehand and saved each other's public key, i.e. Step 1 in Fig. 2.

An authenticator will only authenticate and sign the keys of users who can prove ownership of the key presented. As shown in Fig. 2, C provides A with its public key, a signature on a random string and the random string. Device A uses these values to verify the signature and ensure that C holds the private key corresponding to the public key presented. Before a PA connection can be established, the connection requester must prove ownership of its authentication credentials, as illustrated in Step 3 in Fig. 2. This is similar to how A verifies C in Step 2 but, in addition, C must provide B with its public key signed by A. Thus, B receives the authentication credentials and can verify that the public key of C has been signed by a valid authenticator. If the signed key decrypts to C's public key using A's public key, the process has verified that C is the owner of the signed key and the PA connection request is accepted.

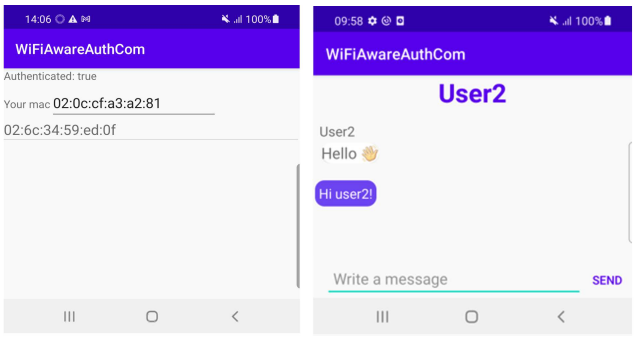
IV. Proof-of-Concept Implementation

This section introduces an implementation of the proposed solution presented in the previous section, which is embedded in a proof-of-concept Instant Messaging (IM) application.

A. Authentication, Certificates and Cryptography

1) The Authentication Server: The authentication of devices was done by manually signing Certificate Signing Request (CSR)s on an Ubuntu machine using the OpenSSL toolkit1. The authentication component was created by generating a private-public Elliptic Curve key pair with a corresponding root certificate. The component worked as a CA responsible for signing CSRs and issuing all of the client/server certificates.

2) Generating Certificates: User credentials were created using the Java Keytool 2 command. The Keytool command is used to store, manipulate and access key pairs and certificates in a Java Keystore. The generated private-public key pair of the user was used to create a keystore and a CSR. Two types of certificates were created: a validated certificate signed by the CA and a self-signed certificate. A valid certificate is generated by providing the CSR to the CA and then using OpenSSL to sign the CSR using the CA's private key. A self-signed certificate is generated by using the client's own private key to sign the CSR. The root and the client certificate are loaded into the client keystore. This process is the same for the CA-signed certificate and the self-signed certificate. The keystore is then manually loaded into the internal file system of each device and used for setting up a TLS connection.



(a) The Main Activity interface. (b) Two users chatting using the chat activity, as seen by `user1`.

Fig. 3. The Instant Messaging application interfaces.

3) Cryptography: ChaCha20 with Poly1305 and Advanced Encryption Standard-128 (AES) in Galois/Counter Mode (GCM) are the cipher suites used for symmetric encryption in the proposed solution. Using ChaCha20 with Poly1305 is fast and battery friendly and is therefore suitable for running in mobile phones. However, AES is still the standard encryption algorithm used and is faster than ChaCha when implemented in a device that includes specialized AES hardware [13]. Both cipher suites used in the proposed solution are Authenticated Encryption with Associated Data (AEAD) ciphers and they have been hard-coded into the application.

B. The IM Application

The application comprises of two graphical user interfaces as shown in Fig. 3: the Main Activity and the Chat Activity. The Main Activity lets the user see nearby peers, and the Chat Activity lets the user chat with peers in the formed cluster.

1) The Main Activity: The Main Activity is the first interface presented to the user and is responsible for handling the Wi-Fi Aware functionality and setting up the application server. The Wi-Fi Aware functionality includes turning on the radio hardware, joining or forming a cluster, establishing a subscribe and publish session, and establishing a data path. Fig. 3(a) displays the interface showing the peers available for communication in the cluster. The authentication status is displayed in the top left corner in Main Activity. Selecting a MAC address from the list will launch the Chat Activity and a TLS connection to that user will be initiated. The owner of the MAC address will receive a notification stating that another user wishes to communicate. Also, the Main Activity handles PA functionality and enables users to request authentication before starting to chat with a peer.

2) The Chat Activity: A TLS connection between two peers is initiated when the chat activity is launched. The device that first launches the activity becomes the client in the connection. If a mutually authenticated connection takes place, messages are exchanged and will appear in chat bubbles, as shown in Fig. 3(b).

3) Peer Authentication: Fig. 4 shows a user with a self-signed certificate that has not yet been verified by a peer. This user can request authentication credentials from a peer

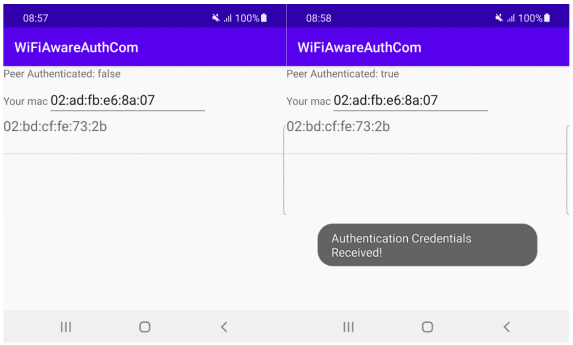


Fig. 4. Status of an unauthenticated user before (left) and after (right) PA.

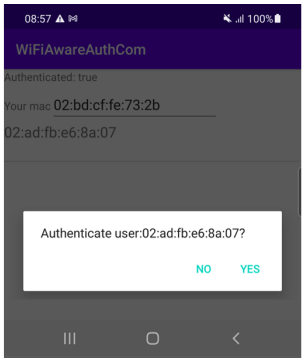


Fig. 5. Option box including the MAC address of the user requesting authentication, presented to the verifier.

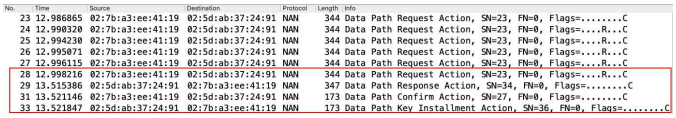


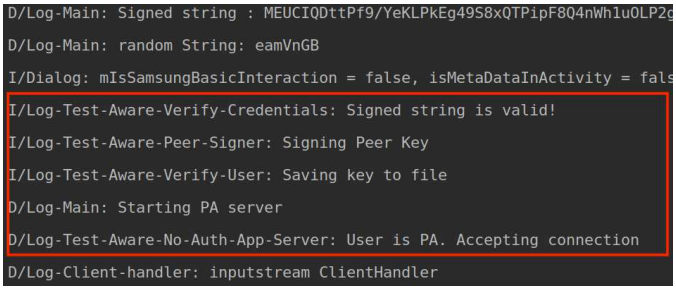
Fig. 6. Wi-Fi Aware handshake showing the data path establishment and the installation of keys.

in the list by clicking on an address. This enables devices to exchange public keys and signed messages in order to receive PA. When an authenticated user receives an authentication request, it is presented with an option box that enables them to decline or accept the request, as shown in Fig. 5. The approaval of the request triggers a PA process and the TLS server is started. As a result, the PA display is changed and the user is notified about the authentication, see Fig. 4.

V. Solution Validation and Evaluation

The proposed solution has been experimentally validated and evaluated in several aspects including discovery and connectivity times, message transmission times for both cryptographic schemes, and peer authentication times.

On the link layer, a WPA2 protected data path with pairwise security association is established during the Wi-Fi Aware 4-way handshake, as specified in [4] and shown in Fig. 6. During the setup, a symmetric key was established from a pre-shared key (PSK) specified during implementation and installed on the devices. After the handshake, the established symmetric key was used in encryption to protect the Data frames and this was verified by evidencing the Data is Protected flag in QoS data frames.



(a) The signature was valid so that the PA server could start and accept the connection.

(b) The PA server did not start because the credentials provided were not signed by a known authenticator.

Fig. 7. Android Studio log showing peer authentication requirement.

A. Transport Layer Security

The link layer alone is not sufficient to provide verification of users' identities because of the lack of authentication mechanisms in Wi-Fi Aware. Authentication of users is provided using the mTLS protocol for CA-authenticated peers and TLS for PA authenticated peers. A high level of security is achieved by forcing every device to use the cipher suites recommended in TLSv1.3 [14] and most recent best practices [15].

We observed that the handshake would fail in case a client uses a TLS version different to the one required by the server. A client or server without a valid certificate in an mTLS connection will also result in the handshake failing. It can therefore be concluded that unauthorized peers are prevented from communicating with users at the transport layer, and that the data being exchanged is both confidential and integrity protected due to the implementation of TLS.

B. Application Layer Security

A client requesting PA credentials or a PA connection is verified in the application layer. Instead of mutual authentication being performed in the transport layer, as in mTLS, the client is authenticated using the cryptographic schemes implemented in the proof-of-concept application. An authenticated peer will only start the PA server if the authentication credentials provided are valid and signed by a known authenticator, as shown in Fig. 7(a). On the contrary, the server is started when incoming connections are requested from the peers that are not verified, as illustrated in Fig. 7(b).

The additional security level implemented in the application behavior ensures that only trusted non-signed up peers can use the network for data exchange.

	Mean	Median	SD
Starting the application	2.46	2.46	0.01
Attach to cluster	0.08	0.09	0.02
Service discovery	0.64	0.62	0.14
Connectivity	2.63	2.53	0.93
Total time	5.81	5.69	0.95

TABLE I
STATISTICAL MEASURES OF WI-FI AWARE SETUP TIMES (SECONDS).

C. Connectivity Time

Two Samsung Galaxy A71 smartphones have been used for the experimental trials. We performed 500 runs in total and the sample application was restarted for each run on both devices.

Table I presents statistical measures of each event. Starting the application includes the time used to set up the SSLContext [1] for each run. The SSLContext handles device keys and the certificate necessary for establishing an mTLS connection. The attach to cluster event includes the time it takes for a device to discover and synchronize to a NAN cluster. The service discovery time is the time used to discover a published service and the connectivity time consists of the time it takes to set up a secure data path between devices from the moment the service is discovered.

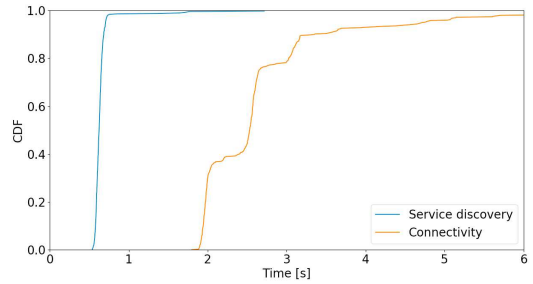
Connectivity is the most time-consuming event with a mean of 2.63 seconds. The mean value exceeded the median, indicating a right skewness in the distribution of connectivity times which include the 4-way handshake process. Observations showed that it could take between 0.22 and 4.75 seconds to perform a handshake. The second most time-consuming event was starting the application, which includes also setting up an SSLContext, with the latter taking around 1.17 seconds. Interestingly, the times to discover a cluster and a published service are close to negligible showing that NAN can be a good solution for dynamic scenarios with high user mobility.

The Cumulative Distribution Function (CDF) in Fig. 8(a) shows that discovery time was almost constant, with the exception of a few large outliers. In addition, there was a low autocorrelation of 0.018 between two subsequent runs, indicating that there is no evident pattern of correlation between runs.

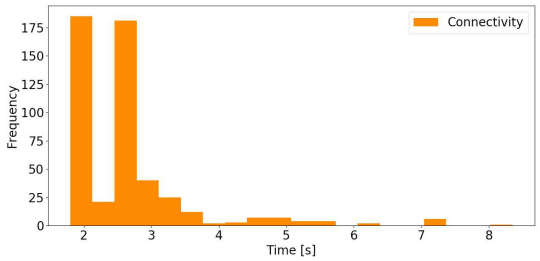
The variation in connectivity times is higher than for discovery times, see Fig. 8(a). This process can take up to 8 seconds and the plot has a few plateaus, which correspond to an aggregate of connection times of around 2 seconds and 2.6 seconds, respectively. Fig. 8(b) confirms this. To investigate whether there is a dependency between the discovery and connectivity times within a run, the correlation between the two data sets was found to be 0.025, which indicates that there is no significant dependency between the events.

A performance comparison between our previous work, utilizing Wi-Fi Direct, and the present is summarized in Table II. Wi-Fi Direct discovery is the time used by a device to discover broadcasts sent by a group, and connectivity is the time used by the same device to join the group. To compare the performance on an equal basis, the Wi-Fi Aware discovery time is a summation of three events: starting the application, attaching to cluster and service discovery.

[1] https://developer.android.com/reference/javax/net/ssl/SSLContext



(a) CDF plot of service discovery and connectivity times.



(b) Histogram of connectivity time.

Fig. 8. Statistical analysis of discovery and connectivity times.

	Wi-Fi Aware			Wi-Fi Direct		
	Mean	Median	SD	Mean	Median	SD
Discovery	3.18	3.16	0.14	3.42	3.07	1.17
Connection	2.63	2.53	0.93	2.03	1.90	0.46
Total time total	5.81	5.69	0.95	5.45	5.00	1.54

TABLE II
COMPARISON OF WI-FI AWARE (LEFT) AND WI-FI DIRECT (RIGHT) DISCOVERY AND CONNECTIVITY DELAY (SECONDS).

On average, the Wi-Fi Aware solution used 0.24 seconds less to discover a service than the Wi-Fi Direct solution. However, Wi-Fi Aware had a longer connectivity delay, making the average total time to set up a connection 0.36 seconds less for Wi-Fi Direct. This difference is probably unnoticeable in terms of user experience and the average total time of the two connectivity technologies compared well.

Sigholt et al. [8] noted that the total time used to discover and connect to a group could take up to 10 seconds in the edge cases. Considerable variation in connectivity times is also present in the Wi-Fi Aware solution, making the total time to establish a network unpredictable.

The devices used in the Wi-Fi Direct solution experienced some difficulties discovering group broadcasts, causing the device to set up a separate group. This issue was not present in the experiment conducted with the Wi-Fi Aware application.

D. Messaging

The time it takes to send messages between the two devices using both AES in GCM and ChaCha20 Poly1305 as cipher suites are also compared. The measuring was carried out using the same devices and the experiment included sending both 3-byte and 32,680-byte long messages. A prerequisite

for the experiment was that an mTLS connection had been established.

32,680 byte Message	Mean	Median	SD
Total time ChaCha20 Poly1305	0.185	0.179	0.021
Total time AES in GCM	0.192	0.180	0.095
3 byte Message	**Mean**	**Median**	**SD**
Total time ChaCha20 Poly1305	0.083	0.072	0.027
Total time AES in GCM	0.097	0.094	0.017

TABLE III
STATISTICS OF THE TIME (SECONDS) IT TAKES TO SEND TWO DIFFERENT MESSAGES WITH AN MTLS CONNECTION.

The time it takes to send long messages using ChaCha20 as an encryption scheme, was slightly less than the time it takes with AES, see Table III. However, a two-sided t-test gives a p-value of 0.45, making the findings statistically insignificant. The average time it takes to send short messages between two devices was also slightly less for ChaCha than for AES. In this case, the two-sided t-test gave a p-value of $7.37 \cdot 10^{-6}$, which is statistically significant given that the mean of ChaCha was 0.083 seconds and of AES was 0.097 seconds. Hence, this confirms the hypotheses that messages sent using ChaCha perform better.

E. Peer Authentication Connectivity

The performance of the DPA scheme was evaluated based on the time it took to receive PA and the time it took to request and start the PA server. Three Samsung Galaxy A71 devices were used in the experiment. The prerequisites for the experiment was that a data path had been established between the communicating devices. For testing purposes, all requests and responses were automated, and no user input was required. All times were recorded on the peer-authenticated device.

Table IV shows statistical summaries calculated from the data collected during the experiment. The verify peer event was measured from the time of the authentication request until the user received the authentication credentials. The average time used, a mean of 1.78 seconds, is significant.

The request connection event is the time it took to request and receive a response that the PA server had started and was ready for communication, as shown in Table IV. The average time used, a mean of 2.32 seconds, is considerable. The event is required every time a peer-authenticated user wants to communicate with another peer and is therefore added to the total connectivity time.

	Mean	Median	SD
Verify Peer	1.78	1.77	0.10
Request Connection	2.32	2.24	0.42

TABLE IV
STATISTICAL SUMMARIES OF THE TIME (SECONDS) USED TO VERIFY A PEER AND REQUEST A PA CONNECTION.

VI. CONCLUSION

This work presented a solution that provides Wi-Fi Aware-enabled secure communication in out-of-coverage scenarios. Wi-Fi Aware enables devices to be aware of published services by other peers and to establish a data path for communication among them. Measures to enable secure and authenticated communication are additionally provided using the mTLS protocol. The solution also provides a decentralized authentication scheme for users who have not signed up for the service before moving into an internet-isolated location. The solution is validated through an instant messaging application which integrates additional security features to offer secure communication with end-to-end encryption between communicating peers, in addition to authentication of users in both online and offline scenarios. The performance and security features have been tested and analyzed. It is observed that Wi-Fi Aware can be a good alternative to many scenarios with high user mobility but ad-hoc designed security mechanisms, such as those proposed in this work, are needed to ensure that critical services adhere to security expectations.

REFERENCES

[1] A. Al-Akkad, C. Raffelsberger, A. Boden, L. Ramirez, A. Zimmermann, and S. Augustin, "Tweeting'when online is off'? opportunistically creating mobile ad-hoc networks in response to disrupted infrastructure." in *ISCRAM*, 2014.

[2] A. Al-Akkad, L. Ramirez, A. Boden, D. Randall, and A. Zimmermann, "Help beacons: Design and evaluation of an ad-hoc lightweight sos system for smartphones," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 1485–1494.

[3] Bluetooth SIG, "Core Specifications 5.3," accessed February 1, 2022. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification/

[4] Wi-Fi Alliance, "Wi-Fi Aware Specification version 3.2," Tech. Rep., 2020.

[5] Global Stats-StatsCounter, "Android Version Market Share Worldwide," https://gs.statcounter.com/os-version-market-share/android, accessed: 2021-12-06.

[6] D. Camps-Mur *et al.*, "Enabling always on service discovery: Wifi neighbor awareness networking," *IEEE Wireless Communications*, vol. 22, no. 2, pp. 118–125, 4 2015.

[7] L. Almon *et al.*, "Desynchronization and MitM Attacks Against Neighbor Awareness Networking Using OpenNAN," in *Proceedings of the 19th ACM International Symposium on Mobility Management and Wireless Access*, 2021, pp. 97–105.

[8] Ø. Sigholt, B. Tola, and Y. Jiang, "Keeping connected when the mobile social network goes offline," in *2019 International conference on wireless and mobile computing, networking and communications (WiMob)*. IEEE, 2019, pp. 59–64.

[9] Elise Smebye and Kirsten Skaug. Source Code. [Online]. Available: https://github.com/elisebsm/AwareAuth

[10] J. Huang and D. Nicol, "An anatomy of trust in public key infrastructure," *International Journal of Critical Infrastructures*, vol. 13, p. 238, 2017.

[11] S. Boeyen, S. Santesson, T. Polk, R. Housley, S. Farrell, and D. Cooper, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008. [Online]. Available: https://rfc-editor.org/rfc/rfc5280.txt

[12] I. Santos-González *et al.*, "Decentralized Authentication for Opportunistic Communications in Disaster Situations," in *Ubiquitous Computing and Ambient Intelligence*, ser. Lecture Notes in Computer Science, vol. 10586. Cham: Springer International Publishing, 2017, pp. 558–569.

[13] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 8439, Jun. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8439.txt

[14] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[15] Y. Sheffer, R. Holz, and P. Saint-Andre, "Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls)," IETF, RFC 7525, 2015.