Encrypting Data to Pervasive Contexts

Hanno Wirtz[†‡*], Torsten Zimmermann[‡*], Matteo Ceriotti[§], Klaus Wehrle[‡]
[†]European Patent Office
[‡]Chair of Communication and Distributed Systems, RWTH Aachen University
[§]Networked Embedded Systems Group, University of Duisburg-Essen
{wirtz, zimmermann, wehrle}@comsys.rwth-aachen.de, matteo.ceriotti@uni-due.de
[*] **Equal Contribution**

***Abstract*—Pervasive applications evolve around the user's context, making it a full-fledged communication party. However, no equivalent approach to End-to-End communication exists that allows a sender to bind data, through encryption, to a target context. Existing solutions require central infrastructures or only apply to the immediate context, restricting pervasive applications.**

We propose Encryption to Trusted Contexts (ETC), a communication security building block that enforces context bindings while preserving the data handling and forwarding mechanisms of the original application. Our approach leverages a Broadcast Encryption scheme to protect data in transit and binds decryption to the verification of an expressive, flexible context specification based on trusted, unforgeable sensing. Data can then be encrypted from and to any context, securely exchanged among devices, and made accessible only within the specified context. We demonstrate the feasibility of ETC in a prototypical implementation for ARM TrustZone devices, realizing communication security for context-driven, pervasive applications.

I. Introduction

Emphasizing the *context of users* [1] over their identity, pervasive applications complement Internet End-to-End (E2E) approaches with an untethered, mobile communication paradigm. Context, as the set or combination of sensor readings, thereby becomes a full-fledged entity with which sending devices can establish End-to-Context (E2C) communication. As an E2C application example, Floating Content [2] binds data to a context defined by an anchor zone and a validity time period. Location-aware mobile devices inside this context replicate and share data via ad-hoc local wireless communication; outside of the context, data is to be deleted. The E2C communication paradigm thereby enables intuitive, context-driven applications.

Application data may then be sent and received directly, forwarded opportunistically [3], or placed and retrieved within the environment [4]. E2C communication security, i.e., a mechanism to encrypt and protect context-bound data in transit and at rest, then becomes an obvious requirement for handling meaningful and sensitive data. For example, data in Floating Content is per design bound to, i.e., only accessible within, a context. However, without E2C encryption, data has to be communicated in clear text, making it accessible in *any* context and effectively removing the context binding. Communication security, equivalent to E2E approaches, thus is a direct feature for the viability and applicability of E2C applications [5]–[12].

However, no *E2C encryption* mechanism exists that allows a) a sender to encrypt, and thereby bind, data to a target context from outside of this context and b) a receiver to decrypt data only if the current context matches the target one. Two fundamental characteristics of E2C communication render the design of such a mechanism challenging. First, contexts cannot, outside of the context, be associated with a (public) key for encryption [13], [14]. Second, the set of recipients that populate the target context in E2C approaches, e.g., a location at a point in time [2], is inherently unknown at encryption time, rendering host-based E2E encryption infeasible [15], [16].

Indeed, E2E encryption requires a defined set of recipient hosts and is thus unable to support recipient sets dynamically formed by contexts. Encrypting data to each individual potential recipient incurs a prohibitive overhead, while group encryption [17] requires key re-distribution when recipients leave the group. Similarly, Attribute- and Identity-based Encryption [18], [19] fail as recipient attributes and identities are unknown at the time of sending. Context-driven approaches either require central infrastructures [20]–[22] or derive encryption keys exclusively between devices in a common context [13], [14]. In this, their applicability is limited to the respective infrastructure or ad-hoc context, in contrast to a building block for distributed E2C communication security. Mitigating these shortcomings, an E2C encryption mechanism would then allow secure, pervasive existence of application data while enforcing context bindings.

In this paper, we thus propose Encryption to Trusted Contexts (ETC), an E2C mechanism that a) leverages the target context specification made by the application to simultaneously define the decryption policy, i.e., indicate the required sensor readings, b) enables the binding, encryption, and transmission of data from any context to the target context, and c) ensures its decryption only within the intended, defined context. Our design builds on three main components, namely a) a specification defining the context that data is associated with and addressed to, b) Broadcast Encryption (BCE) [23] affording encryption via a system-wide public encryption key and decryption using per-user secret keys and c) a Trusted Application (TA) within an ARM Trusted Execution Environment (TEE) that realizes the triple functionality of *sealing* the private BCE decryption key on the device, trusted *sensing* of contexts, and *verifying* the data context policy as the prerequisite for decryption. Our design is motivated by the context awareness of current smartphones as well as the proliferation of ARM TrustZone technology.

ETC departs from existing security approaches (Section II) and addresses the unique threat model of E2C communication (Section III). Building on BCE and a trusted context sensing component (Section IV), ETC encrypts data to sensor-based contexts (Section V). Our prototypical ARM TrustZone implementation shows the feasibility of ETC (Section VI), realizing an E2C communication security building block (Section VII).

II. State of the Art

A. Complementing E2E Security

Disruption-tolerant Networking security [24] offers encryption to context endpoints, using public-key or identity-/attribute-based approaches [25]. In their own right, Identity-based Encryption [18] and Attribute-based Encryption [19] offer encryption on recipient identities or attributes instead of host specifications. A key generator distributes identity or attribute-bound keys and decryption requires the associated key(s). Common to these approaches is the requirement to specify recipients via their attributes and identities at the time of sending. In contrast, we strive for encryption to contexts that accommodate an a priori undefined set of recipients.

B. Trusted Sensing

Multiple approaches [26]–[29] afford trustworthy sensor information by incorporating a trusted component in the sensing and reporting process. This component signs or attests to sensor information and thus affords a measure of trust in the values used in sensing approaches or as a trigger to access pre-installed components, e.g., a trusted drive application [28]. Our use case of distributed pervasive applications requires avoiding attestation to a central service in favor of local trusted sensing.

C. Trust- and Context-driven Security

Multiple approaches, e.g., [16], [20], [30] define context-driven security in the form of smart geographical contexts that adapt the application upon authorizing a user, assuming and restricting to a trusted infrastructure that is deployed beforehand. In contrast, ETC emphasizes infrastructure independence to provide a generic building block for mobile applications.

Mitigating the need for an infrastructure, pairing co-present devices based on the common context, e.g., using observed audio or wireless signals, affords ad-hoc communication security [13], [14]. However, said pairing only affords encryption between devices in the current context and thus does not afford a distributed encryption mechanism that spans multiple contexts, e.g., sending data to a future and/or distant context. As a design alternative, the concept of fuzzy extractors derives a cryptographic key from a context definition, thus affording data encryption bound to a context and subsequent decryption upon attestation of said context [31]. In ETC, we opt to use BCE for both its revocation capabilities and the intuitive restriction of data access within an application by way of secret user keys.

Cesena [32] outlines a preliminary design of combining BCE, to protect data in transit, and trusted attestation of device states. While building on the same components, ETC addresses a fundamentally different problem, namely trusted context sensing as a building block for communication security. We furthermore define a complete architecture and report on a real-life implementation and evaluation of our design.

III. Threat Model

The threat model in our design revolves around unauthorized access to application data, i.e., outside of the respective application or outside of the specified context. The threat of unauthorized access by malicious users, which are not part of the application that data belongs to, or software is inherent to the underlying untrusted, unreliable communication scenario. Similar, intermediate devices or infrastructure elements, which store data, might try to decrypt it, even while adhering to their respective application functionality. We account for this threat of *outside attackers* using Broadcast Encryption (cf. Section V-C).

In contrast, *inside attackers* are participants of the application attempting to access the data in an invalid context, e.g., devices in Floating Content that are outside of the anchor zone and/or validity period. An inside attacker may try to fake the required context as well as manipulate or forge the context specification to achieve an unauthorized decryption. We address this threat by making decryption dependent on trusted sensing and verification of the specified context as well as integrity protection of context specifications (cf. Section V-D).

With regard to the ETC Trusted Application, we follow the standard threat model and assumptions [28]. Namely, we assume that the TA cannot be compromised. Furthermore, physical attacks are out of the scope of our security design.

IV. Background

A BCE system [33] for a number of users N generates in a setup-phase a system-wide public key PK and a secret key SK_i for each user $i \in \{1, ..., N\}$. In order to "broadcast" a message M to a subset S of all users, i.e., $S \subseteq \{1, ..., N\}$, the sender encrypts M to the cipher text C using the public key and indicating S. Notably, the system size N is not static but may be increased [23]. In order to "broadcast" a message M to a subset $S \subseteq \{1, ..., N\}$ of all users, the sender encrypts M to the cipher text C using the public key and indicating S

$$C = Enc(PK, S, M).$$

For performance reasons, the broadcast payload M is actually encrypted using a (random) symmetric key, e.g., using AES, and only said key is encrypted by the BCE system.

A user j receiving an encrypted message C then decrypts it using its secret key, deriving the symmetric key M required to decrypt the accompanying payload M'

$$M = Dec(S, j, SK_j, C, PK).$$

The strength of BCE systems lies in the fact that colluding users, who are legitimately part of the BCE system but are outside of S, cannot retrieve the original message M.

ARM TrustZone [34] offers a Trusted Execution Environment (TEE), the Secure World, on the device, *parallel* to the user-facing Normal World. In contrast to Trusted Platform Module solutions, no single physical entity affords trust in a *Trusted Application* (TA) executing in the TEE. Instead, the TrustZone realizes a lower privilege level to which execution can only be switched using a narrow *Secure Monitor Call* (SMC). Via this call, a Normal World application also exchanges data with a Secure World TA.

ARM TrustZone thereby offers Trusted Platform Module-equivalent services within a full-fledged execution environment that can make use of the device's hardware capabilities instead of being restricted to a single, dedicated hardware module. In light of Trusted Applications that perform complex tasks, such as cryptographic computation, this difference is non-negligible, motivating our Trusted Execution Environment-based design.

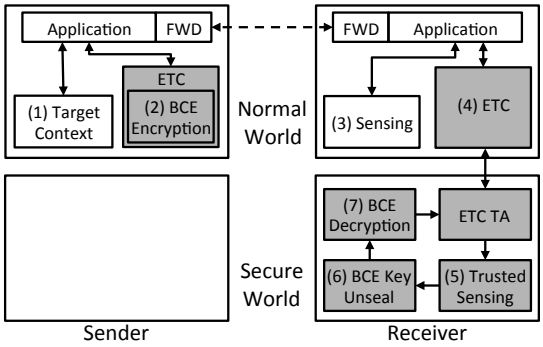


Fig. 1. ETC components (shaded) in the Normal World and Secure World TEE. A legacy application, e.g., Floating Content [2], at the sender specifies a target context and calls ETC for BCE encryption of data before transmission. Receiving devices then trigger ETC decryption if untrusted sensing matches the target context. Actual decryption in the Secure World requires trusted sensing to verify the target context as well as unsealing the BCE secret key. Decrypted data is then passed back to the application in the normal world.

V. Encryption to Trusted Contexts

ETC affords secure data handling on mobile devices in context-driven applications leveraging the device's sensing capabilities. Fig. 1 shows an overview of the functional components and their placement on a device. Please note that, while we design ETC for ARM TrustZone devices, it is applicable to other TEE-approaches, e.g., *Intel TXT* [35]. We thus provide a functional description of our design and refrain from any TEE-specific implementation details.

A device in a legacy application, e.g., Floating Content [2], that leverages our proposed E2C security mechanism, first obtains the public encryption key and a private decryption key in the application-specific Broadcast Encryption (BCE) system. The private key is *sealed* by the ETC TA of the device (not pictured in Fig. 1), hiding it from the user and other software on the device (Section V-A). When sending data, the application defines a target context specification for decryption (step (1) in Fig. 1, Section V-B) and passes data and specification to ETC for encryption (step (2), Section V-C). Sending and intermediate forwarding of encrypted data remain unchanged in the application. Receiving devices trigger decryption, if untrusted Normal World sensing matches the context specification (step (3), V-D1), by passing the encrypted data to the ETC TA in the Secure World (step (4)). There, ETC requires trusted sensing to verify the context specification (step (5), V-D2) before *unsealing* the secret key (step (6)) and decrypting the data (step (7), V-D3).

A. Bootstrapping ETC

Our design builds on fundamental BCE encryption, using a public key, to a defined subset of all possible recipients ($S \subseteq N$). Each recipient then requires a valid secret key for decryption. An E2C-secured application sets up a BCE system [33] with a system-wide public encryption key and secret, private decryption keys. Due to the need of distributing a secret key to each application participant, we envision a trusted online service to afford this one-time setup.

As an initial bootstrapping step, a user contacts the service to obtain the public key and the user-specific secret key. This step comprises a crucial part of our design. Specifically, in order to bind data decryption to trusted context sensing by the TEE, ETC *seals* the secret key for decryption from the user and other software. The secret key is therefore exchanged between the service and a distinct TA on the mobile device realizing a secure key exchange protocol. The TA then encrypts the secret key with, e.g., a unique storage key or a dedicated file encryption key [28]. Sealing the secret decryption key prevents an inside attacker from decrypting data in invalid contexts.

Intuitively, users partake and require E2C security in multiple applications. ETC thus supports the bootstrapping and sealing of multiple application-specific BCE private keys. Applications may then trigger the decryption of their data at the ETC TA, which unseals the respective private key (cf. Section V-D). As a means of authenticating each application, a hash of its code pages present in memory at the Secure Monitor Call, may be compared with a value stored at installation [36]. Such authentication then prevents malicious applications from triggering the decryption of foreign data.

After the bootstrapping phase, a user is able to encrypt application data using the public key. As the main contribution of ETC, this encryption binds data to a context specification.

B. Context Specification

Data may be encrypted to a specific context, i.e., a set of required sensor values. Subsequently, this context specification is passed to ETC. While the specification, in principle, may refer to any type of sensor values, we envision palpable and predefinable sensor information, e.g., location, time, movement, environment, or presence, to play a prominent role as a sender defines the specification outside of the target context. Numerous examples exist for user-defined context definitions that directly facilitate application functionality [2], [5]–[8], [26]–[29].

In the simplest case, a context indicates single sensor values, such as `gps_location=(x,y)`, or a combination of single values. However, context information may contain noise due to sensor inaccuracies, manufacturing differences, calibration errors, etc. A robust context specification thus may define ranges of sensor readings, e.g., `validity_period=(start_time, end_time)`, that make up for such noise. E.g., in Floating Content, a robust context specification would define a buffer around the anchor zone, by adjusting the validity radius or by defining it as a rectangle or a polygon.

Furthermore, a context may be constructed as a series of (combinations of) single sensor readings or ranges, e.g., `loc_series=[(`x_1`,`y_1`),(`x_2`,`y_2`)...(`x_n`,`y_n`)]`. With a series, data may be bound to a geographical path in a location-based game or a series of environment readings, such as Bluetooth Low Energy beacons that carry a BCE-encrypted nonce or respond to a challenge to prove their authenticity.

The format of a specification may take various forms [1], [28], e.g., using a markup language or even a scripting language such as Python. In this, our design supports arbitrary context specifications, as defined by each application that leverages ETC, and realizes the subsequent verification of this specification against trusted sensor information.

C. Encryption

Encryption in ETC builds on the underlying BCE system and thus requires the public key as well as an indication of

the set of recipients S. However, as a notable difference to the conventional usage of BCE, we assume encryption to the full set of application participants, i.e., $S = N$. This is because, in contrast to making the data accessible to only a subset of *recipient identities*, e.g., the set of activated recipient devices, ETC envisions data to be generally *intended* for all possible recipients, i.e., application participants, and binds decryption to the trusted sensing of a specified context.

In order to achieve this binding, ETC encrypts a hash $h(CS)$ of the context specification, concatenated with the data, to cipher text C

$$C = Enc(PK, S, M');\ S = N, M' = M \| h(CS).$$

As such, data is protected against outside attackers that forward or store data without participating in the respective application.

ETC makes use of the context specification in two further aspects. First, it serves as the addressing identifier in the respective mechanism that forwards encrypted data *towards* the target context. Hence, the clear-text context specification, as defined by the legacy application, is sent along with the encrypted data. For example, a context specification in the Floating Content application comprises the anchor zone and the expiration date, enabling intermediate devices to decide whether to forward, store, or discard the data. Second, ETC uses the hash of the context specification to verify the integrity of the clear-text context specification used to trigger decryption.

D. Decryption

A receiving device may decide to decrypt the data if the current context matches the associated, clear-text context specification. In this case, it may *trigger* the ETC TA, initiating *trusted context sensing* and verification as well as data *decryption*.

1) Trigger: Triggering the ETC TA entails a context switch between the Normal World of the regular OS and the Secure World of the TA. As such, it entails a computational and temporal overhead, in addition to the overhead of trusted sensing and, eventually, decryption. In order to preserve system availability and resources, triggering the TA thus should only occur if the associated overhead can be amortized.

To this end, an application matches the clear-text context specification to (untrusted) sensor information obtained in the Normal World. The context sensing step may be part of the application functionality, enabling to "piggyback" the aforementioned comparison, or may be performed dedicatedly. A sensed context matching the specification then triggers the trusted context sensing by passing both the cipher text and the context specification to the ETC TA via the SMC call. We detail this step in the next section.

In combination with *untrusted* context sensing in the Normal World, an inside attacker may easily forge the sensor data indicated in the clear-text context specification to trigger an "unauthorized" context switch to the Secure World. Our design does not address this issue for two reasons. First, we envision ETC to serve as a building block that does not interfere with the original application behavior with regard to sensing and handling of context information as well as decryption requests. Second, (unauthorized) trusted sensing in fact exhausts the resources of the attacker's device, essentially performing a DoS attack. On resource-constrained mobile devices, this provides an incentive to refrain from said unauthorized decryption attempts.

2) Trusted Context Sensing: The trusted sensing component queries all sensors indicated in the clear-text context specifications and verifies whether the sensor values match the specification. Intuitively, single values are valid if they match exactly while any value within a range is considered valid. Similarity metrics [31] such as euclidean distance, hamming distance, or set difference, that define a measure of closeness between sensed and specified values within a metric space, e.g., for (a series of) locations, may further account for valid but noisy sensor information. An invalid reading or missing sensor value, due to a sensing error, excessive noise, or sensor failure, fails the verification process and the sensing component returns an error. A positive verification triggers the actual data decryption. Notably, sensor readings in ETC never leave the TEE, mitigating the need for signatures [26]–[28] or certificates [29] that attest their trustworthiness to an outside entity. In the following, we highlight the two most important design aspects of our context sensing component.

Series of readings: Sensing a context that specifies a series of sensor readings requires the TA to return control to the Normal World between readings to ensure device availability. Hence, the sensing component stores the single readings and matches the evolving partial context against the context specification.

Sensor state: Untrusted software could manipulate the sensor state in the Normal World, damaging the integrity of sensor readings produced after switching to the TEE [28]. The TA thus resets the required sensors to a known, *trusted sensor state* and maintains control until all readings are produced and control is passed to the decryption process.

3) Data Decryption: Triggered by a positive context verification, the TA temporarily *unseals* the secret key SK_i within the TEE and decrypts the cipher text C

$$M' = Dec(S, i, SK_i, C, PK);\ S = N$$

to derive the original data M and hashed context specification

$$M' = M \| h(CS).$$

In order to verify the integrity of the clear-text, untrusted context specification CS', that was passed to the ETC TA from the Normal World, we check whether

$$h(CS') == h(CS).$$

This step serves to prevent any inside attacker from forging the clear-text context description in transit to match any, currently valid context. In case of a positive integrity check, the TA re-seals the secret key and returns the decrypted data M to the calling application in the Normal World via the SMC call.

Notably, the trusted functionality afforded by ETC ends with the return SMC call and no assumption can be made about the trustworthiness of functionality handling the decrypted data in the Normal World. While this is independent of the encryption scheme or availability of a trusted component, it enables the decoupling of data from its context. As a possible solution, ETC could be integrated with a trusted channel between TA and Normal World applications [36], which allows the TA to verify the integrity and state of the calling application. Furthermore,

multiple BCE schemes afford tracing of malicious software to punish such misuse. The tracing entity would interact with the malicious software, derive the secret key(s) used to decrypt data, and revoke the key(s) by removing the associated user(s) from the set of recipients S. User devices may directly perform this functionality or may invoke the BCE system service.

VI. Evaluation

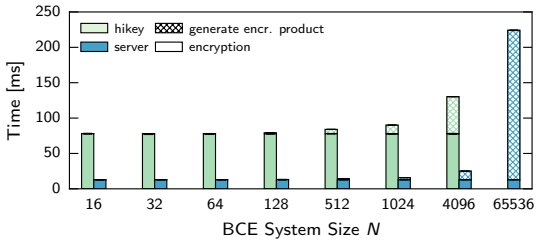
We implemented our ARM TrustZone-driven design of encryption and decryption on trusted contexts on a *HiKey* development board [37] to evaluate the performance of our design. The hardware characteristics of the board (ARM Cortex-A53 CPU at 1.2 GHz, 2 GB RAM) thereby represent the processing capabilities of modern smartphones. We realize the Secure World components (cf. Fig. 1) within the *OP-TEE* [38] Open Source TEE while the board runs Debian 16.06 as the Normal World OS. In order to show trends for large-size systems, we additionally present results gathered using a Ubuntu Desktop 14.04 machine with an Intel i7 2.93 GHz CPU and 4 GB RAM. Finally, we port the *PBC_bce* Broadcast Encryption Library [39] to the TEE to realize the bootstrapping, encryption, and decryption functionality of ETC at the trusted online service, in the Normal World at the sender, and in the Secure World at the recipient, respectively.

Because several approaches address trusted sensing [26]–[29], we do not evaluate sensing performance as the result would be specific to our device, sensor, and driver combination. Functionally, a sensor-specific trusted driver resets and queries the sensor in the Secure World to obtain the required readings. We omit implementation details of our ETC TA, that are specific to our environment of HiKey board and OP-TEE, in favor of reproducible computational performance results. Notably, [28] discusses the performance of trusted GPS sensing.

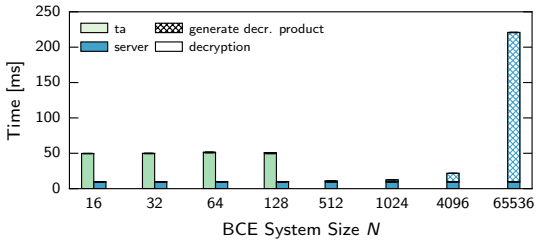
Our evaluation focuses on our main design choice, namely the novel use and cost of BCE encryption and decryption in a trusted environment. Specifically, *system initialization* incurs a substantial overhead, e.g., 321 s using the HiKey board for $N = 4096$ potential recipients. In larger systems, this becomes infeasible on mobile devices, supporting our design of initializing large-size systems at a dedicated service. Furthermore, BCE incurs, at each device, a negligible *space overhead* of $(2N + 1) * 52$ bytes to store the public encryption key and 52 bytes for the unique secret key of the user. In contrast to the aforementioned one-time overheads, the predominating cost of ETC is the recurring *time overhead* of encryption and decryption. Notably, each verification of (complex) context specifications, even under noise, does not affect the decryption overhead, as verification is a sequentially isolated step. In the next section, we thus evaluate the time overhead and contrast the performance of our TEE-based, decentralized solution against two design alternatives: a) encryption and decryption using symmetric keys per user protected by RSA-2048 digital envelopes; b) a central service providing context-bound data upon trusted remote context attestation [26], [27].

A. Time Overhead

We evaluate the time overhead incurred by BCE encryption in ETC. We evaluate decryption time overheads using our current ETC TA prototype implementation, denoted **ta** in all following figures, and encryption time overheads of the **hikey** Normal World component of ETC. In order to show trends for larger N, we use the aforementioned **server** device.



(a) Isolated encryption time overhead.



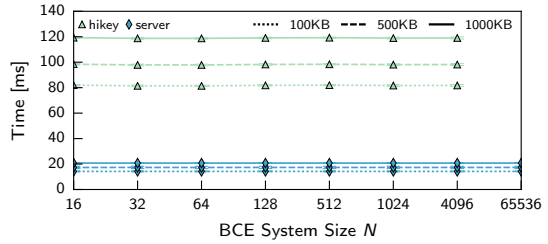
(b) Isolated decryption time overhead.

Fig. 2. Isolated measurements of BCE functionalities show a constant time overhead for encryption and decryption at the recipient, regardless of the BCE system size N. In contrast, re-generating the encryption or decryption product in case of revocation incurs a time overhead proportional to N.

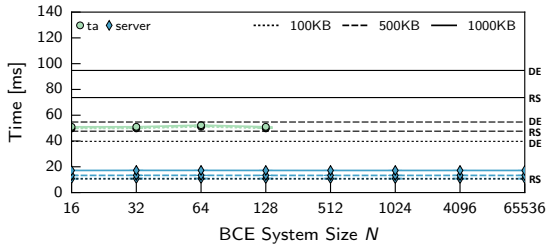
1) Isolated BCE Overhead: We first evaluate the time overhead of BCE encryption and decryption over increasing application sizes N on the HiKey board and the server. Fig. 2 indicates the respective time overhead of encrypting and decrypting a 256 bit AES key (solid bars). We find that, on the HiKey device, the encryption step in the Normal World (Fig. 2(a)) incurs a roughly constant overhead of 78 ms regardless of the recipient set $S = N$. Correspondingly, decryption in the ETC TA (Fig. 2(b)) shows a fixed time overhead of 50 ms for all measured system sizes. The ETC TA thereby makes use of AES hardware support in the HiKey board, outperforming Normal World encryption based on a software-only solution.

Our results display a constant BCE-specific time overhead below 100 ms for encryption and decryption, supporting our design of communication security for fully distributed approaches on mobile devices. Notably, each encryption and decryption step in ETC only requires this fixed time overhead. Re-computing this product incurs a substantial overhead that grows with N and, for larger system sizes, dominates the encryption or decryption overhead (meshed bars in Fig. 2). This overhead occurs during revocation; the removal of the respective index from both products requires their re-calculation.

2) Complete Time Overhead: In this section, we measure the complete time overhead of encryption and decryption, as observed by a Normal World application incorporating ETC, within the Normal World component and ETC TA, respectively, for increasing system sizes N. As in [2], we choose application data of sizes 100 kB, 500 kB, and 1000 kB. For comparison, we measure the overhead incurred by the two design alternatives of 1) AES keys protected by RSA-2048 digital envelopes in the Normal World and 2) a centralized service decrypting and delivering data based on context attestation.



(a) Encryption time overhead as observed by application.



(b) Decryption time overhead as observed by application.

Fig. 3. Encryption and decryption time overhead for different payload sizes over increasing application sizes N, as measured on the HiKey board and the server device. The results denote the time requirements as observed by an application that uses ETC as a building block for communication security. Decryption overhead is depicted in comparison to results for RSA-2048/AES digital envelope (DE) and remote attestation service (RS) design alternatives. We omit encryption results of DE and RS alternatives for readability.

In the former, a sender encrypts data using an AES symmetric key and encrypts that key with the recipient's RSA-2048 public key. The recipient first decrypts the AES key and then decrypts the application data. In the latter design, a symmetric encryption scheme (AES) between a sender, central service, and (mobile) recipient affords binding decryption of data to trusted context attestation. The sender encrypts data to the service and disseminates it along with a target context specification. In this context, a recipient attests the required sensor values to the service. Upon verification, the service decrypts the data and re-encrypts and delivers it to the recipient.

In ETC, the time overhead comprises encrypting the application data and hashed context specification using a symmetric encryption scheme, and the encryption of the random symmetric key using BCE. Our current implementation uses AES in CTR mode for symmetric cryptography. Decryption then comprises the corresponding steps at the recipient. We thus measure the total time overhead for encryption in the Normal World of the HiKey board and decryption, within the ETC TA, for increasing application data sizes of 100 kB, 500 kB, and 1000 kB. Again, we provide measurements using the server machine as an outlook.

Fig. 3(a) depicts the time overhead results of *encrypting* data on the mobile device (HiKey) and the server. We find that encryption is governed by the size of data and ETC achieves encryption times of roughly 80 ms for data sizes of 100 kB (100 ms for 500 kB, 120 ms for 1000 kB), independent of the BCE system size. In comparison to isolated BCE performance (Section VI-A1), these results clearly show the dominating BCE overhead of affording flexible one-to-many encryption. Especially, this shows in the comparison with 6 ms overhead for 100 kB of data (22 ms for 500 kB, 43 ms for 1000 kB) using RSA-2048/AES digital envelopes of single recipients (not shown in the plot). While all approaches show an overhead proportional to the application payload, BCE achieves results that are competitive with less sophisticated schemes.

Correspondingly, Fig. 3(b) illustrates the time overhead of *decrypting* data at the recipient in ETC and the digital envelope and remote service alternatives. Please note that we omit time delays incurred by communication from the results shown for the remote service as these depend on the communication network rather than the respective security functionality.

We find that, for 100 kB and 500 kB, both DE and RS perform comparably or better to our TEE-based design. This is due to the context switch required for BCE decryption functionality in the Secure World. For data sizes of 1000 kB, the additional encryption and decryption steps in RS as well as the Normal World performance of AES in DE incur a greater time overhead than our ETC TA. Indeed, the decryption overhead is roughly constant, as confirmed by the server measurements for larger N. In comparison to the higher overhead of RSA-2048 decryption of an AES key and AES decryption of data in DE, ETC TA benefits from AES hardware support in the ARM TrustZone Trusted Execution Environment.

With regard to isolated BCE functionality (cf. Fig. 2(b)), the overhead is dominated by the (fixed) time required to decrypt the BCE-encrypted AES key. Specifically, owing to the aforementioned AES hardware support, decrypting 100 kB, 500 kB, and 1000 kB of data in the ETC TA only requires 0.2 ms, 0.7 ms, and 1.3 ms, respectively.

In the application of ETC to pervasive applications, we argue that the combination of a constant time overhead incurred by the BCE scheme and hardware-supported decryption of actual application data affords an efficient communication security building block. ETC then extends (or combines) this basic efficiency regarding security with the notion of trusted sensing. In this, ETC constitutes a novel design that simultaneously accounts for the role of context information in pervasive applications, the need for communication security, and the efficiency requirements of battery-driven mobile devices.

VII. Conclusion

We propose Encryption to Trusted Contexts (ETC) for pervasive End-to-Context communication approaches and applications, affording a mechanism for communication security that is equivalent to E2E solutions. ETC leverages Broadcast Encryption to enable data encryption from any application participant and context, and builds on an ARM TrustZone Trusted Application to ensure that decryption of data is dependent on the verification of the context data is bound to. Our design seamlessly integrates in existing approaches, preserving both the communication and data handling as well as the sensing and context processing functionality of the original application. Our implementation of ETC underlines the performance and applicability of our approach on real-world, smartphone-equivalent devices. The result is a fundamental, practical communication security building block that enables a) the handling of meaningful or sensitive data and b) the enforcement of context bindings in pervasive applications.

Acknowledgments

This work has been co-funded by the DFG as part of the CRC 1053 MAKI.

REFERENCES

[1] S. Cho and C. Julien, "Chitchat: Navigating tradeoffs in device-to-device context sharing," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2016.

[2] J. Ott, E. Hyytiä, P. Lassila, T. Vaegs, and J. Kangasharju, "Floating content: Information sharing in urban areas," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2011.

[3] O. R. Helgason, E. A. Yavuz, S. T. Kouyoumdjieva, L. Pajevic, and G. Karlsson, "A mobile peer-to-peer system for opportunistic content-centric networking," in *ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld)*, 2010.

[4] M. Nagy, T. Kärkkäinen, and J. Ott, "Enhancing opportunistic networks with legacy nodes," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 3, 2015.

[5] N. Thompson, R. Crepaldi, and R. Kravets, "Locus: A location-based data overlay for disruption-tolerant networks," in *ACM Workshop on Challenged Networks (CHANTS)*, 2010.

[6] A. A. V. Castro, G. D. M. Serugendo, and D. Konstantas, "Hovering information - self-organising information that finds its own storage," in *IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC)*, 2008.

[7] A. Gupta, M. Miettinen, M. Nagy, N. Asokan, and A. Wetzel, "Peersense: Who is near you?" in *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2012.

[8] E. Miluzzo, M. Papandrea, N. D. Lane, A. M. Sarroff, S. Giordano, and A. T. Campbell, "Tapping into the vibe of the city using vibn, a continuous sensing application for smartphones," in *ACM International Symposium on From Digital Footprints to Social and Community Intelligence (SCI)*, 2011.

[9] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell, "Sensing meets mobile social networks: The design, implementation and evaluation of the cenceme application," in *ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2008.

[10] N. Davies, M. Lau, C. Speed, T. Cherrett, J. Dickinson, and S. Norgate, "Sixth sense transport: Challenges in supporting flexible time travel," in *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2012.

[11] A. Beach, M. Gartrell, X. Xing, R. Han, Q. Lv, S. Mishra, and K. Seada, "Fusing mobile, sensor, and social data to fully enable context-aware computing," in *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2010.

[12] M. Chuah, S. Roy, and I. Stoev, "Secure descriptive message dissemination in dtns," in *ACM International Workshop on Mobile Opportunistic Networking (MobiOpp)*, 2010.

[13] M. Miettinen, N. Asokan, F. Koushanfar, T. D. Nguyen, J. Rios, A.-R. Sadeghi, M. Sobhani, and S. Yellapantula, "I know where you are: Proofs of presence resilient to malicious provers," in *ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, 2015.

[14] M. Miettinen, N. Asokan, T. D. Nguyen, A.-R. Sadeghi, and M. Sobhani, "Context-based zero-interaction pairing and key evolution for advanced personal devices," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[15] Y. Michalevsky, S. Nath, and J. Liu, "Mashable: Mobile applications of secret handshakes over bluetooth le," in *ACM Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2016.

[16] R. Campbell, J. Al-Muhtadi, P. Naldurg, G. Sampemane, and M. Dennis Mickunas, *Towards Security and Privacy for Pervasive Computing*. Springer Berlin Heidelberg, 2003.

[17] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Comput. Surv.*, vol. 35, no. 3, 2003.

[18] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual International Cryptology Conference (CRYPTO)*, 2001.

[19] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy (S&P)*, 2007.

[20] A. J. Lee, J. P. Boyer, C. Drexelius, P. Naldurg, R. L. Hill, and R. H. Campbell, "Supporting dynamically changing authorizations in pervasive communication systems," in *International Conference on Security in Pervasive Computing (SPC)*, 2005.

[21] L. Kagal, T. Finin, and A. Joshi, "Trust-based security in pervasive computing environments," *Computer*, vol. 34, no. 12, 2001.

[22] A. Boukerche and Y. Ren, "A trust-based security system for ubiquitous and pervasive computing environments," *Computer Communications*, vol. 31, no. 18, 2008.

[23] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Annual International Conference on Advances in Cryptology (CRYPTO)*, 2005.

[24] S. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle Security Protocol Specification," RFC 6257 (Experimental), May 2011.

[25] N. Asokan, K. Kostiainen, P. Ginzboorg, J. Ott, and C. Luo, "Applicability of identity-based cryptography for disruption-tolerant networking," in *ACM International MobiSys Workshop on Mobile Opportunistic Networking (MobiOpp)*, 2007.

[26] P. Gilbert, L. P. Cox, J. Jung, and D. Wetherall, "Toward trustworthy mobile sensing," in *ACM Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2010.

[27] A. Dua, N. Bulusu, W.-C. Feng, and W. Hu, "Towards trustworthy participatory sensing," in *USENIX Conference on Hot Topics in Security (HotSec)*, 2009.

[28] H. Liu, S. Saroiu, A. Wolman, and H. Raj, "Software abstractions for trusted sensors," in *ACM International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012.

[29] P. Gilbert, J. Jung, K. Lee, H. Qin, D. Sharkey, A. Sheth, and L. P. Cox, "Youprove: Authenticity and fidelity in mobile sensing," in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011.

[30] C. Li, Y. Zhang, and L. Duan, "Establishing a trusted architecture on pervasive terminals for securing context processing," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2008.

[31] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing*, vol. 38, no. 1, 2008.

[32] E. Cesena, G. Ramunno, and D. Vernizzi, "Towards trusted broadcast encryption," in *IEEE International Conference for Young Computer Scientists (ICYCS)*, 2008.

[33] D. Boneh and B. Waters, "A fully collusion resistant broadcast, trace, and revoke system," in *ACM Conference on Computer and Communications Security (CCS)*, 2006.

[34] ARM, "Building a Secure System usingTrustZone Technology," http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[35] Intel, "Trusted Execution Technology: White Paper," http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf.

[36] J. S. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "Secret: Secure channel between rich execution environment and trusted execution environment," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2015.

[37] 96 Boards, "HiKey Board," http://www.96boards.org/product/hikey/.

[38] "OP-TEE Trusted OS," https://github.com/OP-TEE/optee_os.

[39] B. Lynn, "PBC_bce Broadcast Encryption Library," https://crypto.stanford.edu/pbc/bce/.