Multiply, Divide, and Conquer

Making Fully Decentralised Access Control a Reality

Bernd Prünster[1,2][0000−0001−7902−0087], Dominik Ziegler[3][0000−0002−5930−8216], and Gerald Palfinger[1,2][0000−0001−6633−858X]

[1] A-SIT Secure Information Technology Center Austria
{bernd.pruenster,gerald.palfinger}@a-sit.at
https://www.a-sit.at
[2] Institute of Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
https://www.iaik.tugraz.at
[3] Know-Center GmbH, Graz, Austria
dominik.ziegler@tugraz.at

Abstract. This paper tackles the issue of access control in fully decentralised systems. Previously, access control always fell back to some degree of centralisation. Our work approaches this problem by outsourcing access policy evaluation to the millions of trusted computing bases already deployed in the form of current Android devices. This assures correct policy evaluation to both data owners and those seeking data access. In essence, our solution encrypts to-be-shared data, splits and wraps the encryption key, and cryptographically binds it to an access policy. Policies are evaluated by freely selectable evaluators, that do not need to be enrolled beforehand. Evaluators then interface with attribute providers during policy evaluation. Each evaluator independently reaches a conclusion about whether or not to grant access, leading to a decision by majority vote. We designed this system with practicality and real-world applicability in mind, meaning that it can be deployed and used today. We achieve this by relying on efficient primitives and foregoing expensive cryptographic constructions, making it possible to define even highly complex access policies. Overall, this presents a clear advantage over previous concepts.

Keywords: Decentralised Access Control, Trusted Computing, Peer-to-Peer

1 Introduction

In the past years, decentralisation has been gaining momentum, with cryptocurrencies pushing decentralised payment systems, and projects like *IPFS*[4] advocating decentralised applications. When hosting some service with or without the help of such frameworks, enforcing access policies is trivial, as every access is

[4] https://ipfs.io/

handled by the entity providing the service. New, previously hardly addressed issues arise, however, when the entity owning some to-be-protected data might not be available when this data is to be accessed. Most importantly, when advocating the advantages of decentralised systems, falling back to centralised approaches diminishes some (if not all) of the inherent advantages of this paradigm. We therefore firmly believe that fine-grained access control needs to be made available in a decentralised manner.

We strive for attribute-based access control (ABAC) in this context because this flexible approach can emulate all other access control schemes. Recent advancements in implementing and deploying trusted computing at large have put the realisation of fully decentralised ABAC into reach. The key distinction of our approach compared to previously theorised concepts is its feasibility under real-world conditions.

Contribution: We propose *Multiply, Divide, and Conquer* to enable data owners to outsource data protected by access policies without the need to be involved in the data access and policy evaluation process. Since an open, fully decentralised context without any form of common governance implies mutually distrusting parties, we explicitly target this setting. We cater to this scenario by delegating policy evaluation to trusted computing devices. Our design targets practical applicability in the real world and does not represent a purely academic scheme. To accomplish this, we propose to utilise recent Android devices, equipped with trusted hardware modules, capable of extensive remote attestation capabilities for the policy evaluation process. As we will argue, it is feasible to delegate policy enforcement to current Android devices outside a data owner's control and guarantee correct policy enforcement both to the data owner as well as to the party seeking to access the policy-protected data. As we aim for practicality, our scheme is designed to remain efficient even when using complex policies. We have implemented a prototype and evaluated its performance to show that this goal was reached in practice.

Our design combines established ABAC frameworks with the trusted computing capabilities of modern Android devices and linear secret sharing. The decentralisation aspect is upheld by letting data owners freely select instances for policy evaluation as well as arbitrary attribute providers. In addition, relying on trusted computing for policy evaluation assures all parties that policies are evaluated correctly and honestly.
To securely share some data, the data owner defines an access policy and attribute providers. Data is encrypted, the encryption key is split, and each share is encrypted for a single policy evaluator selected by the data owner. Data, policy, and encrypted key shares can then be published. Upon data access, policy evaluators interface with attribute providers for policy evaluation. Each evaluator decides individually whether or not to grant access by divulging its respective key share to the accessing entity. In essence, if evaluators agree[5] to grant access, the data encryption key can be recovered and data access is thereby granted.

[5] No coordination between evaluators is needed.

2 Background

Multiply, Divide, and Conquer implements fully decentralised ABAC based on trusted computing and secret sharing. This section therefore provides the necessary background on these topics, with a focus on trusted computing on mobile devices.

2.1 Secret Sharing

The idea behind generic (k, n) threshold secret sharing is to split up a secret D among n parties, such that D can be efficiently reconstructed from k shares, while "knowledge of any $k-1$ or fewer D_i pieces leaves D completely undetermined" [14]. More precisely, we rely on Shamir's secret sharing, since it (1) scales well (with the total size of all shares being $k \cdot \texttt{sizeof}(D)$), (2) can be implemented efficiently, and (3) reconstruction does not involve the dealer (who created the shares). As we will argue in Section 4, our design requires no verifiability with respect to the secret sharing scheme.

2.2 Attribute-Based Access Control

Attribute-based access control (ABAC) [4] provides authorisation mechanisms at a granular level. It achieves access authorisation by evaluating user-, global environment-, resource- as well as action-attributes to reach a decision. A widely used industrial standard is the eXtensible Access Control Markup Language (XACML; [8]), which is one of the most frequently referenced works of generic ABAC models. It does not, however, provide a standardised way to ensure data confidentiality. Attribute-based encryption (ABE), on the other hand, cryptographically enforces access control. Some ABE concepts offer properties similar to our proposal as outlined in Section 6.

2.3 Mobile Trusted Computing

Fides [10] is a scheme to verify the integrity of a remote application using the attestation capabilities of Android. It requires an Android device with trusted cryptographic hardware and remote attestation capabilities as supported by devices launched with Android 8.0 or later. The application attempting to prove its integrity first requests the creation of a public/private keypair from the hardware-based keystore and an attestation certificate. This certificate contains hardware-enforced information about the verified boot state and the lock state of the bootloader, certifying OS integrity and verified boot state. In addition, the hardware-enforced values include the OS version and its patch level, which makes it possible to lock out vulnerable smartphones. Furthermore, the attestation certificate contains information about the application requesting the attestation. By validating the signature certificate of the application, Fides can thus verify that the examined application has not been tampered with.

Prünster et al. [9] show that it is possible to utilise the aforementioned concept in a fully decentralised peer-to-peer (P2P) context to thwart Sybil attacks. The approach takes advantage of the fact that Android's remote attestation can be verified completely offline, by checking against Googles hardware attestation root certificate[6]. Although access to device identifiers as proposed by Prünster et al. [9] has been restricted in recent Android versions[7], it is still possible to impose a limit of running only a single instance of an application per device: App developers can enforce that only the so-called *system user* is able to use their app[8].

3 Multiply, Divide, and Conquer

Multiply, Divide, and Conquer requires no set of predefined entities for evaluating and enforcing access policies and no trusted setup phase. Instead, policy evaluation and enforcement needs to be delegated to trusted devices.

We define a current Android smartphone as described in Section 2.3 as *sufficiently trusted*: It is possible to remotely verify a device's integrity using key attestation, when not considering root exploits. Actually compromising an Android device using an exploit takes considerable effort even for device owners. For example, basic local privilege escalation exploits are priced at USD > 100.000[9] as of June 2020. To further raise the bar for attackers, we also introduce basic multi-party computation (MPC) to the policy evaluation process by letting multiple devices evaluate the same policy. The overall evaluation result is then reached by majority vote.

3.1 Architecture

Multiply, Divide, and Conquer is underpinned by a decentralised P2P network where every participant is equal. It employs attestation-based identifiers as discussed in Section 2.3. Our system defines the following actors to enable data sharing with outsourced policy enforcement:

Data owner The *data owner* seeks to publish data and protects it using an access policy. Every data owner is identified using a public-private key pair.
Attribute provider *Attribute providers* are used to attest that a user holds some set of attributes. We assume honest but curious attribute providers. Attribute providers are identified by a certificate.
Storage provider Any *storage provider* of choice may be used to host data. The only requirement towards the storage provider is availability.

[6] `https://developer.android.com/training/articles/security-key-attestation#root_certificate`
[7] `https://developer.android.com/about/versions/10/privacy/`
[8] `https://developer.android.com/reference/android/os/UserManager#isSystemUser()`
[9] `https://zerodium.com/program.html`

Evaluator Policy *evaluator*s are responsible for evaluating and enforcing access policies. Data owners choose a subset of all *reasonably trusted* devices known at policy creation time. This set is composed of Android smartphones whose integrity can be remotely verified using key attestation (see Section 2.3). Consequently, each evaluator's identity is derived from a certificate chain whose leaf is an attestation certificate bound to a public-private key pair and a hardware identifier (see [9]).

Accessor The set of *accessor*s encompasses everyone who wants to access data published within Multiply, Divide, and Conquer. Each accessor is identified by a public-private key pair.

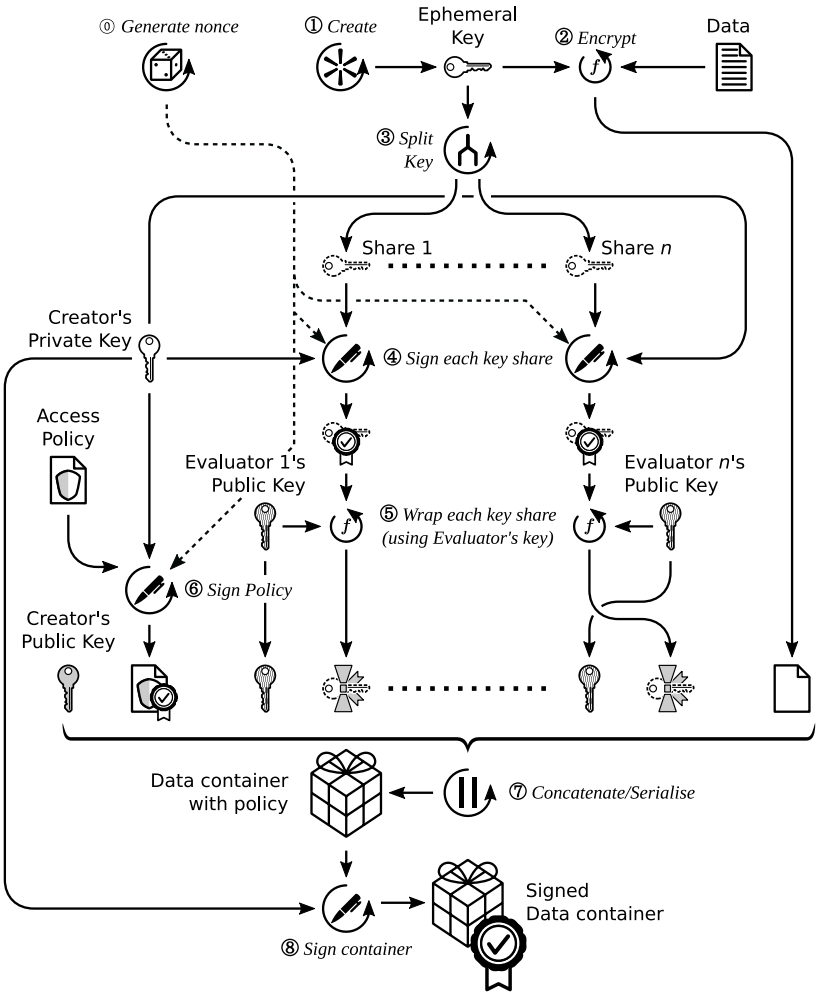


Fig. 1: Encryption process – all components are cryptographically linked.

3.2 Data Publishing

This section presents the general idea behind data encryption, with a detailed process description provided below in accordance with Fig. 1. This process focuses on the cryptographic operations and therefore assumes some data to be protected by an already defined policy at hand:

⓪ The *data owner* creates a nonce.
① The *data owner* creates an ephemeral bulk encryption key and
② encrypts some data using this key.
③ The *data owner* splits the key using a (k, n) threshold secret sharing scheme into n shares. Depending on the choice of k and n, more or less *evaluators* need to be online during policy evaluation (see Section 3.3). The requirement for a majority vote can be implemented by choosing $k > n/2$.
④ The *data owner* attaches the nonce to each key share. It signs this tuple using the private key tied to its identity to create a binding between key shares and data owner.
⑤ The *data owner* wraps each share for a single evaluator using the public key tied to the respective evaluator's identity.
⑥ The *data owner* attaches the *nonce to the policy*. It signs this tuple using the private key tied to its identity to create a binding between policy, shares, and data owner.
⑦ The *data owner* concatenates and serialises:
  - Data owner's public key
  - Signed policy
  - Evaluator's identities
  - Encrypted, wrapped key shares
  - Encrypted data
⑧ The *data owner* signs the resulting data container using the private key tied to its identity. Once the signed data container is published, the data owner can go offline as accessors only need to contact evaluators to gain data access.

3.3 Data Access

Accessing data involves the *storage provider*, *accessor*, *attribute providers*, and *evaluators* and consists of the following steps:

① The *accessor* fetches the data container,
② checks the signature, and
③ extracts the signed policy and all signed, wrapped key shares.
`For each i-th key share:`
  ❶ The *accessor* transmits the signed policy, the data owner's identity and the i-th wrapped key share to the i-th *evaluator*.
  ❷ The *evaluator* checks the policy signature,

❸ evaluates the policy, and requests any information (such as user attributes) from the *accessor* that may be required to reach a verdict about whether access should be granted. This step usually involves one or more *attribute providers*, which is omitted from the process description for the sake of clarity.
❹ In case access should be granted, the i-th *evaluator* unwraps the i-th key share and checks its signature.
❺ The *evaluator* checks whether the received share and the policy belong to each other by comparing the nonce values and whether both items were signed by the same data owner.
❻ If so, the *evaluator* signs the unwrapped share, and
❼ transmits it to the *accessor*.
❽ The *accessor* checks both unwrapped key's signatures (the original one created by the *data owner*, and the one created by the i-th *evaluator*).

④ The *accessor* recovers the bulk encryption key from all received shares, and
⑤ finally decrypts the data.

As can be observed, no actor needs to perform computationally expensive operations. Policy evaluation itself is implemented using XACML, which has been shown to perform well, even on lower-end hardware [15]. A thorough performance analysis is provided in Section 5, which supports the claims of efficiency and real-world applicability.

4 Security Characteristics

This section argues how Multiply, Divide, and Conquer achieves fully decentralised, practically secure, and correct access control based on the data access process described in Section 3.3. Consequently, the security properties of each step are discussed below. Afterwards, general security properties, benefits over the current state-of-the-art, and a sketch of an adversary model are provided.

According to Section 3.3, the *accessor* ① initially fetches an encrypted data container and ② checks its signature using the included public key. Since the *data owner* created this container, we can assume that their correct public key was included. As a consequence, the *accessor* can unconditionally verify the data container's integrity without the need for verifiable secret sharing.
After ③ taking apart the data container and ❶ distributing the signed policy (including the nonce), data owner identity, and wrapped, signed key shares to *evaluators*, ❷ each *evaluator* can validate integrity and origin of the policy by means of a simple signature check.
Assuming policy evaluation (❸) results in the *accessor* being granted access—a conclusion each *evaluator* reaches independently of others—each *evaluator* can then ❹ unwrap their key share to recover the nonce and the signed, plain key share.
By ❺ verifying the share's signature against the previously received *data owner* identity, each *evaluator* can validate the received key share's integrity and origin. By also comparing the nonce attached to the signed policy with the nonce

attached to the key share, the *evaluator* can verify that the key share actually belongs to the policy and has not been replaced with another policy signed by the same evaluator[10].
After ❻ signing and ❼ transmitting their share to the *accessor*, the evaluator is not involved anymore.
The *accessor* ❽ checks both signatures of the received shares (one initially created by the *data owner*, and one created by the *evaluator*). The original signature serves the purpose of enabling the *accessor* to verify the integrity of the received share. Thus no verifiable secret sharing scheme is required. The second signature serves accountability, in the unlikely event that of an evaluator being compromised using a root exploit and not producing the correct key share. Once all this has been done, the *accessor* can ④ recover the bulk encryption key and ⑤ finally recover the data. Appendix A provides more details on the adversary model and discusses the system's general security properties in detail.

While outsourcing a sensitive task like access control to essentially unknown entities may seem irresponsible, in this case it actually provides advantages even compared to traditional client-sever setups. As subsumed in Section 2.3, it is indeed possible to deploy code to current Android devices and remotely verify that it will be executed without modifications and that no sensitive data can be extracted. This effectively reduces the attack surface to highly targeted attacks, which we consider out of scope. In addition, it assures *data owners* and *accessors*, that access policies are indeed correctly evaluated.

5 Implementation and Performance

Multiply, Divide, and Conquer has been implemented in Kotlin, for easy benchmarking on the desktop and on Android. To obtain meaningful results regarding our system's performance, we only evaluated *scheme-specific* operations and do not consider *generic* operations that would be required for any form of confidential data exchange. Bulk encryption, for example, is considered a generic operation and has thus not been benchmarked.

5.1 Implementation Details and General System Characteristics

Multiply, Divide, and Conquer uses SHA3-256 and the Elliptic Curve Digital Signature Algorithm (ECDSA) (curve: *secp256k1*) as signature algorithm. Our system heavily relies on cryptographic signatures, which has two implications: (1) asymmetric operations (and hashing) will account for the bulk of performance overhead and (2) no verifiable secret sharing scheme is required. Instead, an implementation of Shamir's secret sharing[11] is used. Wrapping (encrypting) shares for evaluators relies on the Elliptic Curve Integrated Encryption Scheme (ECIES).

[10] Technically, this could also be accomplished by incorporating the random value used for secret sharing into the policy.
[11] `https://github.com/codahale/shamir`

Creation and verification of signatures, participating in the secret sharing scheme, and policy evaluation are considered scheme-specific operations. These operations are not independent, however. Most prominently, the chosen number of evaluators directly impacts performance on all levels:

- Complexity of the secret sharing scheme increases.
- As each share needs to be signed, more shares result in more signature and verification operations.
- As each shares need to be wrapped, more shares result in more hybrid encryption operations.
- The number of shares increases the overall container size, which results in more hashing operations being performed as part of container signing/verifying.

The remaining factors impacting overall performance are payload and policy size as this affects the number of hashing operations during creating/verifying the policy's signature. Consequently, this also has a bearing on the number of hashing operations required for creating/verifying the overall container signature. Naturally, policy complexity has a direct impact on policy evaluation performance. We therefore used fixed container and policy sizes, as well as a fixed policy complexity.

5.2 Benchmarks

We have split our benchmarks into three groups to reflect the operations carried out by each actor and organised this section accordingly. To provide a constant container and policy size, we used a fixed 1kB policy and a 10MB payload. Policy evaluation was benchmarked against the `IID302` policy from the XACML 3.0 conformance test suite provided by AT&T[12]. This policy was modified to grant access instead of denying it.

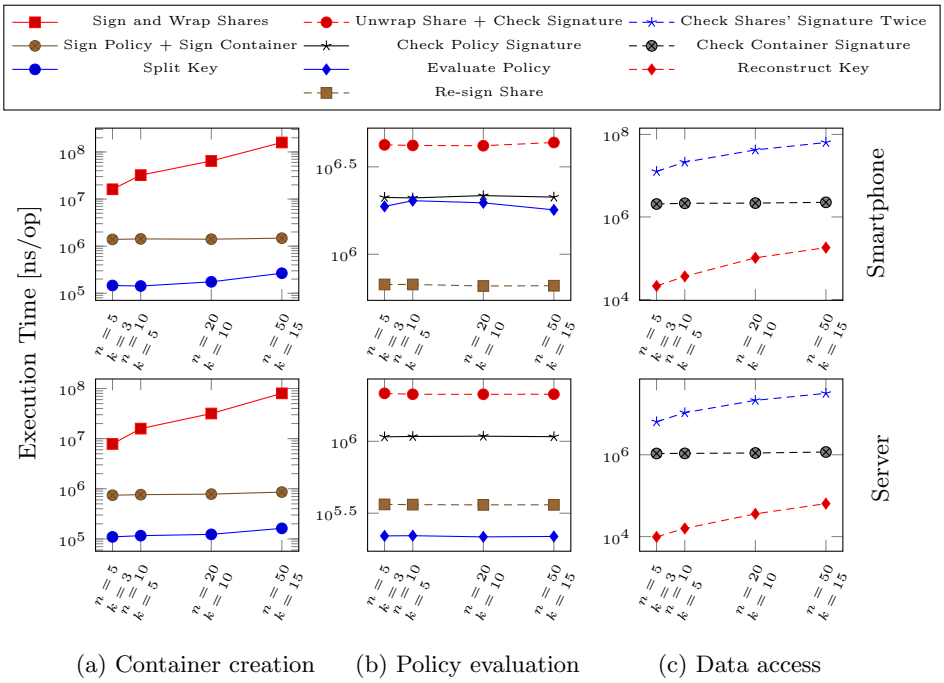
Setup: The primary device used for testing was a *bq Aquaris X*. All benchmarks were run single-threaded, as this reflects assigning only limited resources (i.e. a single CPU core) to Multiply, Divide, and Conquer running in the background. The remainder of this sections discusses the obtained results.

Data Publishing (Container Creation): This task is performed by the *data owner* and has been described in Section 3.2. The following operations as shown in Fig. 1 have been benchmarked:

- Splitting the bulk encryption key among evaluators (Step ③)
- Signing (Step ④) each key share and wrapping (Step ⑤) each share for it's designated evaluator
- Signing an access policy (Step ⑥) and signing the overall data container (Step ⑧)

[12] `https://lists.oasis-open.org/archives/xacml-comment/201404/msg00001.html`

The results shown in Figure 3a confirm the intuitive notion that asymmetric operations are the dominating factor with respect to the overall performance of data container creation. However, at less than 0.1s for realistic secret sharing parameters, overall performance remains satisfactory.



(a) Container creation (b) Policy evaluation (c) Data access

Fig. 2: Execution times: Each operation was carried out 100 times (excluding warmup runs). Server hardware: Intel *Xeon E5-2699 v4* @ 2.20GHz. Smartphone hardware: Qualcomm *Snapdragon™ 626* octa core platform @ 2.20GHz.

Policy Evaluation: This task has been benchmarked from a single *evaluator*'s point of view, since it scales linearly with the number of evaluators. However, this usually happens in parallel from the *accessor*'s point-of-view, which means the figures plotted below can be factored into the overall performance as-is. The following scheme-specific operations as defined in Section 3.3 have been benchmarked:

- Checking a policy's signature (Step ❷)
- Policy evaluation (Step ❸)
- Unwrapping the evaluator's share and verifying its signature (Steps ❹ and ❺)
- Re-signing the decrypted share using the evaluator's key (Step ❻)

Fig. 3b clearly shows that the operations carried out by *evaluators* perform well, taking only few milliseconds even on less-than-current smartphone hardware.

Data Access: The following scheme-specific operations as shown in Section 3.3 were benchmarked from the *accessor*'s point of view:

- Check container signature (Step ②)
- Checking a share's original signature as well as checking the signature produced by the evaluator (Step ❽). These two operations are carried out k times each (since k out of n shares are required to reconstruct the secret)
- Reconstructing the bulk encryption key from k shares (Step ④)

Fig. 3c displaying the *accessor*'s performance clearly correlates with Fig. 3a, since the operations related to secret sharing are again the dominating factor. Given that all scheme-specific operations total to less than half a second even when having to combine 15 shares to recover the key, interacting with *attribute providers* and network latency will have the most impact on the overall round-trip time. In summary, the benchmark figures obtained clearly show that Multiply, Divide, and Conquer performs well and does not strain the targeted smartphone hardware.

6 Related Work

On the surface, Multiply, Divide, and Conquer provides a similar feature set to some ABE schemes. Even though trusted computing has been proposed for access control before, no decentralised approach has so far been brought forward. We therefore refrain from discussing proposals that rely on a central trusted computing instance. Instead, this section focuses on summarising the major contributions in the field of ABE and compares them to our work.

Attribute-based encryption (ABE), in contrast to attribute-based access control (see Section 2), ensures fine-grained access control on a cryptographic level. It defines the recipients of a message as a set of attributes. Access rights are determined based on access structures, either embedded in the key (Key-Policy Attribute-Based Encryption (KP-ABE; [3]) or the ciphertext (Ciphertext-Policy Attribute-Based Encryption (CP-ABE; [1])). Users can only decrypt a ciphertext if their attributes match the embedded policy.

Since its introduction, subsequent work has improved ABE in terms of performance or functionality. The area most similar to our work is multi-authority ABE. First described by Chase [2], multi-authority ABE schemes allow for any party in the system to become an attribute authority. It is collusion-resistant, when assuming a trusted authority in place. Whenever new attribute authorities join, the system key needs to be modified and propagated to all users. Our scheme, on the other hand, imposes no limitation whatsoever on attribute providers, both in overall number and dynamics over time.

Another approach to decentralise ABE was proposed by Müller et al. [7]. They present a distributed ABE scheme, where an arbitrary number of parties

can maintain attributes and secret keys. The scheme eliminates the need for a central attribute authority, but requires a trusted *master* to distributes secret keys Policies are restricted to boolean formulas in disjunctive normal form (DNF) over attributes and their values. We, however, rely on a combination of secret sharing to distribute keys and trusted computing to enforce policies. Thus, our scheme does not rely on a central authority and can also enforce considerably more expressive policies.

Lewko and Waters [6] proposed a similar system to ours. In their scheme, any party, too, can become an attribute authority. Like our work, the proposed scheme does not require any central authority. However, the system has several limitations which impact usability in resource-constrained environments. First, the system is designed around composite order bilinear groups. As was shown by Kiraz and Uzunkol [5], this decision influences key size and thus performance. To maintain a desired security level, key sizes need to be increased. Secondly, the scheme requires a trusted global setup phase. In contrast, our scheme supports environments with mutually distrusting parties and needs no global setup phase.

Summarising, multi-authority ABE offers a cryptographic solution to provide fine-grained access control in distributed environments. More recent variants of multi-authority ABE improve existing schemes by adding support for, e.g., arbitrary large attribute strings [12], revocation [11], hidden policies [16] or tackling efficiency in resource-constrained environments [13]. What almost all variants of multi-authority ABE schemes have in common, though, is that they rely on expensive bilinear pairing operations or on one or more central entities.

7 Conclusions

This paper presented Multiply, Divide, and Conquer, a system to enforce access control in fully decentralised environments based on secret sharing and the trusted computing capabilities of current Android devices. Compared to previously proposed concepts, it relies on fast primitives and traditional trusted computing concepts, thus allowing for complex access policies, while remaining efficient, compared to attribute-based encryption. At its core, it utilises the fact that it is possible to remotely attest the integrity of current Android devices running in unmanaged environments and deploy software to these devices, such that the software cannot be tampered with in an undetectable manner. At no point does our system require any central instance. As such, our solution can be deployed to fully decentralised P2P networks without falling back to centralisation. Previously, this has not been possible. Most importantly, our system has been designed with practicality in mind, meaning that it is usable in the real world and does not remain a purely academic experiment. Its security parameters can be tweaked to suit different needs, all while keeping real-world applicability in mind.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: 2007 IEEE Symposium on Security and Privacy (SP '07), pp. 321–334 (2007), https://doi.org/10.1109/SP.2007.11
2. Chase, M.: Multi-authority Attribute Based Encryption. In: Theory of Cryptography, pp. 515–534, Springer Berlin Heidelberg (2007), https://doi.org/10.1007/978-3-540-70936-7_28
3. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data. In: ACM CCS, pp. 89–98, ACM, New York, NY, USA (2006), https://doi.org/10.1145/1180405.1180418
4. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. Tech. rep., National Institute of Standards and Technology, Gaithersburg, MD (2014), https://doi.org/10.6028/NIST.SP.800-162
5. Kiraz, M.S., Uzunkol, O.: Still Wrong Use of Pairings in Cryptography. CoRR (2016), URL `http://arxiv.org/abs/1603.02826`
6. Lewko, A., Waters, B.: Decentralizing Attribute-Based Encryption. In: Advances in Cryptology – EUROCRYPT 2011, pp. 568–588, Springer Berlin Heidelberg (2011), https://doi.org/10.1007/978-3-642-20465-4_31
7. Müller, S., Katzenbeisser, S., Eckert, C.: Distributed Attribute-Based Encryption. In: ICISC 2008, pp. 20–36, Springer Berlin Heidelberg (2009), https://doi.org/10.1007/978-3-642-00730-9_2
8. OASIS Standard: eXtensible Access Control Markup Language (XACML) Version 3.0. Tech. rep. (2013), URL `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`
9. Prünster, B., Fasllija, E., Mocher, D.: Master of puppets: Trusting silicon in the fight for practical security in fully decentralised peer-to-peer networks. In: ICETE 2019 - Volume 2: SECRYPT, pp. 252–259 (2019)
10. Prünster, B., Palfinger, G., Kollmann, C.: Fides: Unleashing the full potential of remote attestation. In: ICETE 2019 - Volume 2: SECRYPT, pp. 314–321 (2019)
11. Qian, H., Li, J., Zhang, Y., Han, J.: Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation. International Journal of Information Security **14**(6), 487–497 (2015), https://doi.org/10.1007/s10207-014-0270-9
12. Rouselakis, Y., Waters, B.: Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption. In: Financial Cryptography and Data Security, pp. 315–332, Springer Berlin Heidelberg (2015)
13. Sandor, V.K.A., Lin, Y., Li, X., Lin, F., Zhang, S.: Efficient decentralized multi-authority attribute based encryption for mobile cloud data storage. Journal of Network and Computer Applications **129**, 25–36 (2019), https://doi.org/10.1016/j.jnca.2019.01.003
14. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979), https://doi.org/10.1145/359168.359176
15. Suzic, B., Prünster, B., Ziegler, D., Marsalek, A., Reiter, A.: Balancing utility and security: Securing cloud federations of public entities. In: OTM Confederated International Conferences, pp. 943 – 961, Lecture Notes in Computer Science (LNCS), Springer International Publishing AG, Switzerland (2016)
16. Zhong, H., Zhu, W., Xu, Y., Cui, J.: Multi-authority attribute-based encryption access control scheme with policy hidden for cloud storage. Soft Computing **22**(1), 243–251 (jan 2018), ISSN 1433-7479, https://doi.org/10.1007/s00500-016-2330-8

A Security Model

This appendix discusses the security properties of Multiply, Divide, and Conquer. First, assets to be protected are identified and characterised based on the system description provided in Section 3. Next, security services that need to pe provided are derived. Afterwards we define each actor and an according adversary model. Finally, this is mapped against our design's characteristics to show which properties and strategies ensure that each security service can be provided. We assume that all cryptographic primitives perform as expected and consider implementations flaws and targeted attacks out of scope.

A.1 Assets

The assets identified below are directly derived from Multiply, Divide, and Conquer's goals and the workflows described in Section 3.

Payload should only be revealed to those parties, who are rightfully allowed to access it according to the access policy. Every payload is encrypted, signed and cryptographically linked to an associated access policy.

Access policy defines which parties are authorised to access the payload. Access policies are public, must not be manipulated, and are cryptographically signed.

Key shares must only be revealed to authorised parties (according to the access policy) such that those parties are able to decrypt the payload.

Attributes are used to describe any party seeking to access some encrypted payload as inputs for policy evaluation.

A.2 Security Services and Goals

Based on our scheme's goals for fully decentralised access control to securely share data, the following security services need to be provided.

Confidentiality. Assets: *payload, key shares*: A payload shall only be disclosed to accessors who fulfil the access policy associated with a payload. As the payload is encrypted and the encryption key is split, key shares must also be kept confidential and only be revealed to authorised accessors.

Integrity. Assets: *payload, key shares, policy, attributes*: Information must remain intact; modifications must be detected. This excludes manipulated information that remains intelligible (see unforgeability).

Authenticity. Assets: *payload, key shares, policy, attributes*: It must be possible to verify the origin and integrity of any asset, thus authenticating all data.

Accountability. Assets: *payload, key shares, policy, attributes*: Following authenticity, the origin of every asset must be traceable such that the actor can be held accountable. This includes any (legitimate) manipulations of an asset, such as the decryption of key shares by evaluators.

Non-repudiation. Assets: *payload, key shares, policy, attributes*: Following accountability, no actor must be able to unrightfully disclaim authorship (or legitimate manipulation) of any asset.

Unforgeability. Assets: *payload, key shares, policy, attributes*: Forging a policy, payload, or key shares would mean that an attacker could replace assets with a modified, still intelligible version and integrity and signature checks, as well as authenticated encryption schemes would still produce valid outputs.

Availability. Assets: *payload, key shares, policy, attributes*: All information (and the actors required to process it) must be available when needed.

A.3 Actor and Adversary Model

This section briefly formalises Multiply, Divide, and Conquer's actor and adversary model and illustrates how an adversary may try to weaken its security properties by trying to compromise one of is security services. Due to the definition of system actors, attackers inherently assume the role of an adversarial actor. For example, a third party seeking to gain unauthorised access to protected data is considered to be an adversarial accessor.

Data owners seek to publish some data and protect it using an access policy.

Adversarial data owners may produce data that either overburdens *evaluators* with complex policies, or may occupy all other actors with a policy that requires long-running interactions between *accessors*, *attribute providers*, and *evaluators* while consuming *storage providers*' space. In general, limiting policy complexity and introducing timeouts are apt measures to combat adversarial *data owners*. We therefore consider this issue out of scope.

Attribute providers are used to attest that an *accessor* holds some set of attributes—assumed to be chosen such that they behave honestly but curious.

Adversarial attribute providers may, in theory, misbehave by inspecting an access policy to produce attributes accordingly in order to tilt the policy evaluation result towards denial or access. This targets confidentiality or availability. In established schemes like *OpenID Connect*[13], the service provider is free to choose identity providers (IdPs) they trust. Our design takes up on this idea and lets *data owners* freely choose *attribute providers* they trust, without any form of central governance. This strategy is therefore no worse than the current state-of-the-art. As a consequence, an adversarial attribute provider is degraded to an honest, but curious one, if we consider that trusted *attribute providers* behave honestly. The specific trust model to be used can be chosen based on the deployment scenario.

Storage providers are used to host data and are chosen by *data owners*.

Adversarial storage providers may try to gain access to the payload (targeting confidentiality), modify it (targeting integrity, authenticity, unforgeability), replace access policies (targeting integrity, authenticity, unforgeability, accountability). In addition, adversarial storage providers may simply delete data they are supposed to store, thus targeting availability.

Evaluators are responsible for evaluating and enforcing access policies. The

[13] `https://openid.net/connect/`

evaluation process is carried out inside a trusted computing base. Our security model assumes that this secure execution environment can only be compromised by a powerful attacker (see Section 4), which we consider out of scope.
Adversarial evaluators may go offline at any point in time, based on realistic user behaviour. Thus, only availability concerns remain.

Accessors want to access data published within Multiply, Divide, and Conquer.
Adversarial accessors may seek to gain unauthorised access encrypted payloads, thus targeting confidentiality.

A.4 System Properties Providing Security Services

Property 1 (Confidentiality). Assets: *payload, key shares.*
Since means for transport security like Transport Layer Security (TLS) are available, we assume authenticated, encrypted communication channels between all actors. This protects attributes and key shares in transit. The payload is encrypted prior to publishing and the randomly generated bulk encryption key is split and wrapped for evaluators, making these two assets initially confidential. Colluding evaluators could recover the key. However, this would violate the postulated trusted computing properties.

Property 2 (Integrity, Authenticity, Accountability, Non-repudiation). Assets: *payload, key shares, policy, attributes*
Considering honest, but curious attribute providers and TLS-secured connections, provided attributes are authentic. This implies that their source can be verified. As our system employs ECIES, and key shares are signed by the data owner and (after policy evaluation) also by evaluators, their authenticity can be verified. Authenticated encryption is used to encrypt the payload, and polices are signed and cryptographically linked to key shares and payload, the integrity and authenticity of these assets can also be verified at any given point. The use of cryptographic signatures and cryptographic linking using a nonce directly implies accountability and provides non-repudiation.

Property 3 (Unforgeability). Assets: *payload, key shares, policy, attributes*
Assuming all cryptographic primitives perform as expected, forging any asset is infeasible. Considering honest but curious attribute providers and TLS, attributes cannot be tampered with. We assume a state-of-the-art protocol towards attribute providers that ensures freshness of all produced data.

Property 4 (Availability). Assets: *payload, key shares, policy, attributes*
Cheap (and even free), cloud-based storage is abundant. Since payload, key shares, and policy are stored together, this applies to all three of these assets. Defining multiple attribute providers (catering towards redundancy) is possible. In addition, secret sharing parameters can be tweaked to account for *evaluators* going offline to ensure availability.

Assuming the postulated trusted computing guarantees are upheld and attribute providers behave in an honest but curious manner, our design delivers on its promises.