Protecting Documents with Sticky Policies and Identity-Based Encryption

Grzegorz Spyra
The Cyber Academy
Edinburgh Napier University
Edinburgh, UK
g.spyra@napier.ac.uk

Prof William J Buchanan
The Cyber Academy
Edinburgh Napier University
Edinburgh, UK
w.buchanan@napier.ac.uk

***Abstract*—Documents are increasingly being held in public cloud-based systems, and there thus increasingly exposed to accesses from malicious entities. This paper focuses on the integration of sticky policies that are embedded into OOXML (Open Office XML) protecting each of the elements of a data package. Along with this it combines with Identity-Based Encryption (IBE) to securely attach the sticky policy onto data.**

Keywords — OOXML, Sticky Policies, IBE, IRM, Cloud Security

I. Introduction

There is an increasing move towards Cloud-based systems to store data [1], but the methods that we have used in the past are increasing irrelevant. The focus in applying security within Cloud-based systems has typically focused on enhancement of access control methods or on encrypting documents. A major problem, though, is that there is not much in the way of frameworks, which can embed access rights into ordinary documents.

Shamir defined a public key encryption scheme [2], where it is possible to build a secure construction where communicating parties could use a simple text as a public key without a need for keys exchange. Based on this assumption Dan Boneh and Matthew Franklin [3] constructed an efficient fully functional Identity-Based Encryption (IBE) scheme using Pairing-Based Cryptography (PBC).

With the IBE scheme, a message sender can take any arbitrary text known to receiver and use it as a public key. Whereas plain text does not require any further cryptographic safeguards a message receiver requires authentication that proves ownership of the public key, i.e. email address. Using email address as a public key helps communicating parties to share information about Trust Authority (TA) what constrains the key domain for pairing operations. While in proposed construction sticky policies act as document identity giving it TA and access control context.

By selecting from a list of registered TAs, Alice selects preferred Trust Authority (TA), and then receives a template of possible policy rules – predefined access rights within a given security context. After defining policy access rules, the policy set is extended by data owner – Alice rights, and together with document global unique identifier and a TA reference the sticky policy is ready to protect the document.

Section II introduces sticky polices concept and section III describes how to securely keep policy attached to a data. Section IV shows original approach of granular OOXML document access control with preselected XACML policy profiles. Section V briefly describes how using IBE primitive sticky policy can follow data in public cloud. Section VI evaluates IBE in compare to RSA for sticky policy as well as sticky policy itself as an access control method.

II. Sticky policies authorization

Sticky policies carry authorization information required to protect the data. Policy evaluation upon access request can check *who you are*, *what you have*, *what you know*, *where you are* and *when and how you can access* the data. E.g. in countries that adopted OECD data protection directives [4] owner consent related to data access can be represented as an access rule and combined into a policy set. As mentioned before data access can be constrained by time. E.g. sticky policy added to a financial report would define any subject rights to process the report within a defined time slot and before or after a specific date.

```
<Policy>
   <Rule Effect="Permit">
      <Target>
         <Subject "GROUP(BusinessEngineering):{956EFF…}"/>
         <Resource "TA_URI/{8AA1F374-FAX1-4E5D-
BDF1…}"/>
         <Action "Read"/>
      </Target>
   </Rule>
</Policy>
```

Fig. 1. XACML Rule example

XACML access request construct represents access tuples, with subject, object and predicate. Subject is the data owner or data processor who wish to access the object. Object is the resource document that can be represented by cloud data hosting provider path and unique data identifier. Predicate defines an action that subject is entitled to base on the policy rules. Because of its internal XML structure XACML policies are defined via attributes represented by name/value pairs. XACML sticky policy subject can be constrained by a technical Role [5] represented as a group in a target system, where e.g. Role is equal «BusinessEngineering». Because sticky policy remains unencrypted its attribute values could be anonymized as a further safeguard. «BusinessEngineering» Role could be represented by a global unique identifier (see Fig. 1) from within given Trust Authority context.

XACML policy model simply combines Rules, Policies and PolicySets into Policies or PolicySets (see Fig. 2) to protect the resource and enforce access rights defined by data owner. Possibility of Policy and PolicySet nesting gives many possibilities to represent access conditions.

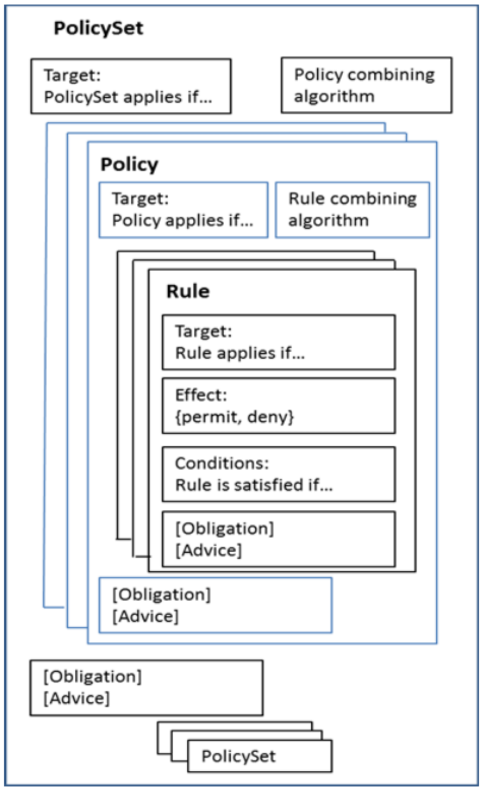

PolicySet
Target: PolicySet applies if...
Policy combining algorithm
Policy
Target: Policy applies if...
Rule combining algorithm
Rule
Target: Rule applies if...
Effect: {permit, deny}
Conditions: Rule is satisfied if...
[Obligation] [Advice]
[Obligation] [Advice]
[Obligation] [Advice]
PolicySet

Fig. 2. XACML Policy Construct [6]

Interesting functional part that is defined by XACML are obligations and advice. Obligation is a *must* requirement compared to non-obligatory advice, which *can* be considered during access control decision. Obligation is a directive specifying obligatory operation after access request decision. E.g. obligation can instruct to raise a security incident after Eve was denied access to the data. Advice can instruct Bob to use his academic email identity because he does not have a valid educational *ac.uk* domain address. Important feature of both obligation as well as advice is the fact that these can enforce data re-encryption under larger key space or even different cryptographic method.

III. Sticky policies IBE authenticity

The policy which is stuck to the data (Fig. 3) cannot be tampered by an illegitimate person. Acting as a public key the sticky policy is authenticated by IBE scheme. Only the exact key can be used to decrypt the cipher-text. IBE is a public key asymmetric cryptographic primitive therefore for a given public key encrypting the message exists one private key decrypting the cipher-text with this message. If an attacker would try to change the sticky policy attached to the data in this construct after TA authorizes falsified request the received private key cannot be used to decrypt the cipher-text.

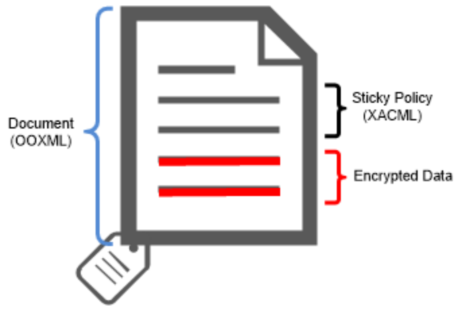


Fig. 3. IBE secured OOXML document with Sticky-policy attached

In addition, the model can provide data non-repudiation assurance with an extra cryptographic operation. Authenticated Identity-Based Encryption (Authenticated IBE) delivers both message confidentiality and non-repudiation on top of IBE scheme [7]. To implement this authorship safeguard either sticky policy or OOXML document meta-data should carry information about the data owner. Sender i.e. Alice using own private key can authenticate the encryption.

If data integrity is required there are existing Identity Based Signature (IBS) schemes [8]. This safeguard is more expensive than non-repudiation as requires separate encryption and signing operations, while Authenticated IBE is even faster than actual IBE encryption. Considering other available technologies for integrity and non-repudiation Blockchain might be preferred. It verifies data in a historical context [9] and Blockchain service integrated with Trust Authority (TA) may govern any illegitimate re-encryption attempt of the amended data. Changed document despite of initially defined sticky policy rights giving only Read rights can be rejected by TA therefore change will not be added to the block chain.

Sticky-policy integrity can also be checked under Authenticated Identity-Based Encryption [10] scheme albeit it requires policy private key to be leased by the Trust Authority (TA) during initial encryption.

IV. Embedded Access Rights

Office Open XML (OOXML) that was combined with XACML policy is represented as related parts gathered into container called package. Package is an ordinary ZIP file containing content-type item, relationship items and parts [11]. OOXML can represent documents with underlying meta-data using WordprocessingML subclause. Workbooks use dedicated SpreadsheetML data format. PresentationML can store rich-presentation meta-data and finally DrawingML specifying images location and appearance within a package.

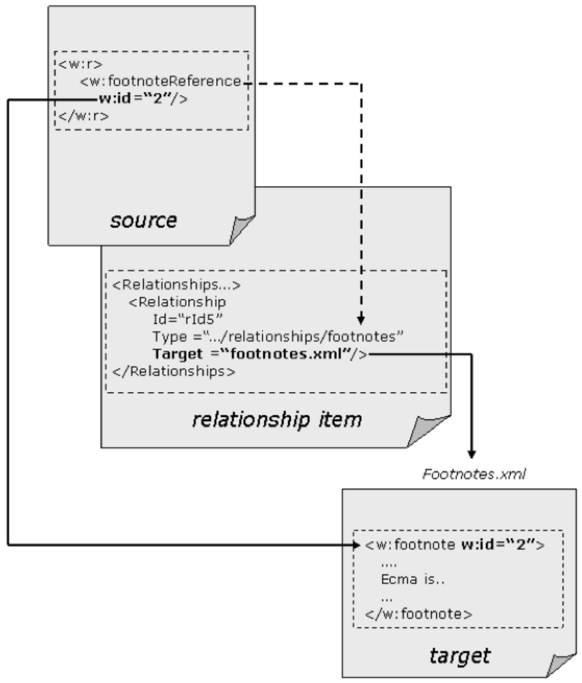


Fig. 4. OOXML internal Implicit relationship [11]

One of the possible granular access control implementations can leverage OOXML implicit relationships (see Fig. 4) that describe references from document parts to other package resources and combine them with XACML Hierarchical [12] and Multiple Decision [13] profiles. Functionality allowing efficient expression of a policy constraint that will apply to an entire OOXML document hierarchy, rather than having to specify a separate constraint for each document element, simplifies policy definition therefore reduces risk that access controls are evaluated correctly. Furthermore Multiple Decision profile allows combinations of multiple access control decisions where single access request can evaluate access rule for more than one resource. While *resource-id* represents part of the hierarchy i.e. package, where access rules are applied, the internal OOXML package referenced elements identifiers (See Fig. 4) are anchors for XACML sticky policy.

Such approach not only controls who can access the data but also what part of a document can be accessed. Of course OOXML editor application so called *consumer* and *producer* requires additional safeguards to edit document with granular access control applied but yet OOXML already defines basic access control attributes that can be leveraged to interpret XACML rights.

V. IBE WITH STICKY POLICY AS IDENTITY

To illustrate how the document and sticky policy can be cryptographically protected, Alice encrypts a document using IBE BF [3]. This requires the setup of a policy key public Q_{POL}:

$$Q_{POL} = H_1(POL_{ID}), \tag{1}$$

where $H_1$is a hash function defined on group $\mathbb{G}_1$ of prime order q such as $H_1: \{0,1\}^* \longrightarrow \mathbb{G}_1^*$, which maps sticky policy POL_{ID} into a single point on an elliptic curve.

Alice then generates a random number r from group $\mathbb{Z}_q = \{0, \dots, q-1\}$ under modulo q and calculates the parameters:

$$\begin{cases} U = rP \\ V = m \oplus H_2\left(\hat{e}\left(R_{pkg}, rQ_{POL}\right)\right) \end{cases}, \tag{2}$$

where V is derived from a symmetric $\oplus$ operation function over message m and bilinear map ê. Secret key as per IBE is computed from bilinear mapping ê where ê: $\mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$.

U and V are then stored inside the OOXML document wrapper, along with the embedded sticky policy.

If Bob wants to access the document, sends an access request and the policy to the Trust Authority (TA) using TA reference from the sticky policy. If he has rights, the TA uses secret master key s and computes private key for given sticky policy as follows:

$$S_{POL} = sQ_{POL}, s \in \mathbb{Z}_q \tag{4}$$

Next TA sends policy response together with sticky policy private key S_{POL} to Bob (Figure 2). Bob can now use symmetric operation $\oplus$ on parameter V and hash function $H_2: \mathbb{G}_2 \rightarrow \{0,1\}^n$ and decrypt the document as follows:

$$m = V \oplus H_2\left(\hat{e}(U, S_{POL})\right) \tag{5}$$

Access right specific decision is made by policy framework based on policy response details, however all possible permissions are interpreted as Read or Read/Write rights.

VI. EVALUATION

Prototyped sticky policies of size between 4 [KB] and 5 [KB] was used to protect the document, which was encrypted using IBE Boneh and Franklin (BF) and AES 256. Furthermore IBE performance was compared to other more legacy RSA encryption - the same public key cryptographic model that Microsoft used for RMSOnline Rights Management System (RMS) [14]. In presented model sticky policy is used to generate a secret key under IBE for AES encryption of the data part. In MS RMS the AES secret key for data part encryption is generated separately and together with policy to follow the data it is encrypted using RSA and then attached to the encrypted data. Therefore, here evaluation looks only into the initial process of policy setup including AES key protection without actual data encryption (i.e. AES 256).

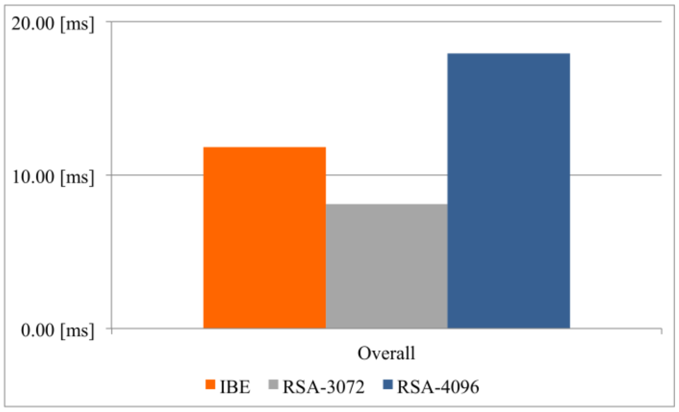


Fig. 5. Times of Sticky policy mapping into AES key space using IBE-BF compared to 3072 and 4096 RSA operations applied to pseudo random AES 256 key [10]

Results show (see Fig. 5) that RSA with key size 4096 requires more time than Pairing Based Cryptography, i.e. IBE to pair XACML policy of size between 4[KB] and 5[KB] into AES key space. RSA 3072 performs better and requires less time to complete cryptographic operations, however soon it might need to be replaced with RSA 4096. Individual tests also show that RSA performed better during encryption compared to IBE pairing. RSA decryption however performed much slower, whereas IBE completes within similar time as in previous pairing with public test. Note that in this scenario RSA has to encrypt not only symmetric key but also access policy, therefore overall performance of RSA 4096 might be comparable to IBE. Finally, evaluation shows that IBE used in our construct performed well compare to RSA even though it did not calculate RMS policy encryption under RSA.

Next evaluation relies on basic sticky policy assumption. Sticky-policies can utilize existing policy frameworks, however an advantage of comprising both a policy and an object (resource) into sticky policies model over keeping the policy separate from the object like in Discretionary Access Control (DAC) model it is its reduced number of model entities and increased DB access performance (see Fig. 6). Having two policy implementations based on transactional databases it is easy to derive query time t_p assuming it is equal to natural logarithm of total records number. In policy-based access control model implementation database maintains not only document information, which is claimed by the subject but it also holds access policies. Policy can store document location information, however in access scenario subject claims resource (document) based on resource information before this policy is evaluated. One can calculate query time t_p assuming we have to query policy each out of p policies and each document out of n documents separately as in

$$\forall a = p \times n \tag{6}$$

$$t_p = \ln(a) = ln(n) + \ln(p)$$

In sticky-policy model the policy is attached to the resource and both are retrieved in one single request. We can calculate query time based on a single table query, assuming policy is encapsulated with a document and both are stored together, as in

$$\forall\, p = n \tag{7}$$

$$t_s = ln(n)$$

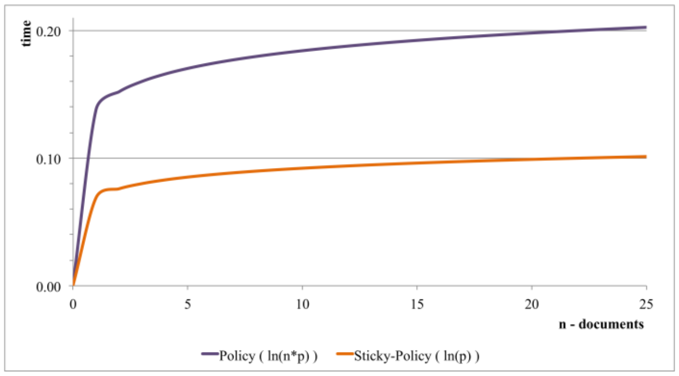
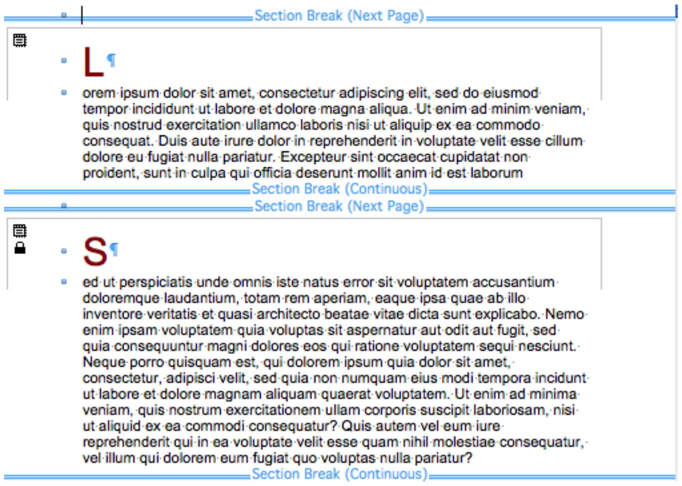


Fig. 6. DAC Policy (tp) and Sticky-Policy (ts) DB queries response time [10]

Finally the last OOXML and XACML evaluation using explicit relationship and master document model [11] from WordprocessingML subclause shown that described here granular access control method can be used to control access to sub-documents. Whereas policy access response denied resource Write access model added Read-Only attribute to master document what was represented as a padlock on the document outline (see Fig. 7).



Section Break (Next Page)

L¶

orem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

Section Break (Continuous)

Section Break (Next Page)

S¶

ed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

Section Break (Continuous)

Fig. 7. Sticky policy applied to master document

Granular access model built for this evaluation is just a proof of concept as various sources discourage using master document model due to several integrity problems with complex documentation.

VII. Conclusions

There is a lack of methods, which can be used to control the access to data elements within documents, thus sticky policies can be used to protect restricted elements within documents.

References

[1] J. Luna, N. Suri, M. Iorga, and A. Karmel, “Leveraging the Potential of Cloud Security Service-Level Agreements through Standards,” *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 32–40, 2015.

[2] A. Shamir, “Identity-Based Cryptosystems and Signature Schemes,” in *Advances in Cryptology*, vol. 196, G. R. Blakley and D. Chaum,

Eds. Springer Berlin Heidelberg, 1985, pp. 47–53.

[3] D. Boneh and M. Franklin, "Identity-Based Encryption from the Weil Pairing," *SIAM J. Comput.*, vol. 32, no. 3, pp. 586–615, 2003.

[4] OECD, "Recommendation of the Council concerning Guidelines governing the Protection of Privacy and Transborder Flows of Personal Data (2013)." pp. 11–37, 2013.

[5] A. Anderson, "XACML Profile for Role Based Access Control (RBAC), Version 2.0," 2004.

[6] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)," in *ABAC '16 Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control Pages 13-24*, 2016, pp. 13–24.

[7] B. Lynn, "Authenticated Identity-Based Encryption," 2002.

[8] J. C. Cha and J. H. Cheon, "An Identity-Based Signature from Gap Diffie-Hellman Groups," *Int. Assoc. Cryptologic Res.*, pp. 18–30, 2002.

[9] K. Okupski, "Bitcoin Developer Reference," Eindhoven, The Netherlands, 2015.

[10] G. Spyra, W. J. Buchanan, and E. Ekonomou, "Sticky policy enabled authenticated OOXML," in *SAI Computing Conference 2016*, 2016.

[11] Apple, Barclays Capital, BP, The British Library, Essilor, Intel, Microsoft, NextPage, Novell, Statoil, Toshiba, and the United States Library of Congress, "Information technology — Document description and processing languages — Office Open XML File Formats —Part 1: Fundamentals and Markup Language Reference," vol. 2012. ISO/IEC, Geneva, p. 5030, 2012.

[12] B. Parducci, H. Lockhart, E. Rissanen, and R. Levinson, "XACML v3.0 Hierarchical Resource Profile Version 1.0," 2014.

[13] B. Parducci, H. Lockhart, and E. Rissanen, "XACML v3.0 Multiple Decision Profile Version 1.0," 2010.

[14] Sergey Simakov, M. Sieber, and M. Norden, *Azure RMS Security Evaluation Guide*. Microsoft, 2015.