

# In-app Cryptographically-Enforced Selective Access Control for Microsoft Office and Similar Platforms (Regular Paper)

Karim Eldefrawy<sup>1</sup>, Tancrede Lepoint<sup>2\*</sup>, and Laura Tam<sup>1</sup>

<sup>1</sup> SRI International  
333 Ravenswood Ave, Menlo Park, CA 94025  
{karim.eldefrawy,laura.tam}@sri.com

<sup>2</sup> No affiliation  
crypto@tancre.de

**Abstract.** The interplay between cryptography and access control has been widely investigated in the literature. On the bright side, attribute-based encryption (ABE) has appeared as a major cryptographic tool going beyond the all-or-nothing approach of public-key encryption by supporting fine-grained access control for encrypted data. Unfortunately, the deployment and adoption of ABE have been slow, and few commercial widely-used products use it to date. In particular, selective and fine-grained control over what is shared, and with whom, is absent from common data products and formats, such as those generated by commercial products (Microsoft Word documents, Excel spreadsheets, PowerPoint slides, and so on). This lack of selective and fine-grained control results in users simply *not sharing*. This major usability shortcoming impacts defense and military coalition operations, as well as commercial settings, such as life sciences, healthcare, and the financial sectors.

This paper addresses this identified usability problem head-on by proposing a cryptographically-enforced selective access control in Microsoft Office products and similar platforms. We focus on Excel as an illustrative use-case, but note that our work is applicable to (and is implemented for) other Microsoft products such as Word, PowerPoint, and Outlook. Using the JavaScript API for Microsoft Office, we designed and developed simple add-ins that enable cell encryption according to a policy, and requires a key that embeds attributes satisfying the policy in order to decrypt. Our performance evaluation not only shows that cryptographic-based selective sharing of information in widely-deployed and widely-used commercial authoring and collaboration platforms is possible, but also efficient.

## 1 Introduction

Private data sharing remains as of today a critical challenge for individuals, enterprises, and (inter)national organizations, and governments. While sharing data is essential, sharing sensitive data with the wrong entity can have devastating consequences or even be prohibited [2, Sec. 1201]. On the bright side, the literature is rich with cryptographically-enforced access control solutions. A cryptographic tool that naturally lands itself to fine-grained access control is that of attribute-based encryption (ABE) [25]. Here, ciphertexts and keys are associated with attributes which determine when decryption is possible. In a ciphertext-policy ABE (CP-ABE) [13], keys are associated with attributes like `'(continent = Europe) (trust = 2) (org = NATO)'`, while ciphertexts are associated with access policies such as `'((continent == Europe) AND (org == NATO)) OR (trust > 3)'`. Decryption is only possible when the key attributes satisfy the policy.

Using the previous example, note that the encrypter does not need to know the exact identities of all other entities who should be able to access the data, but rather described them in term of descriptive attributes. Would they be issued keys, countries like France, Italy, or Belgium would be able to decrypt, while Sweden or Finland would not (as they are not members of NATO) unless their key embeds an attribute `trust` larger or equal to 4.

Over the past decade, ABE has led to a bounty of applications, from network privacy to health record access-control and secure messaging. While companies like Zeutro [8] starts investigating the use of ABE in Cloud applications, to date the deployment of ABE has been slow. This can be explained by a variety of reasons. Amongst them, (1) it became clear that ABE schemes need to *readily* accommodate new roles, attributes, and access policies to be used; and (2)

---

\* Work performed while at SRI International.

real-world applications of ABE require strong security guarantees under realistic and natural attack models. Thankfully, these initial concerns are no more. The first requirement has been achieved in 2011 by Lewko and Waters [21] in what is called *unbounded ABE* (that is, an ABE that is not bounded in the number of attributes it can handle). Since then, unbounded ABE constructions have been widely improved and made efficient [20,23,24,11,19,12,18,9,15]. As for the security concern, recent ABE schemes are based on well-understood security assumptions against active adversaries [14,9,15], and rely on asymmetric prime-order (Type-III) pairings, the recommended choice by cryptography experts [17].

Fine-grained access control today is not achieved by cryptographic means. When selective access control is available (e.g., the privacy controls proposed by Facebook, or even in online Excel workbooks [7]), it relies on a (replicated) trusted centralized system that shares with a recipient the data she is authorized to see. To selectively share information that is contained in the most used document formats without a centralized system (e.g., `docx` for text, `jpeg` for images, `xlsx` for spreadsheets, or `pptx` for presentations), the commonly used process is to *manually* remove sensitive information over multiple versions of the same file according to the recipient. In our extensive discussion with military departments and agencies, we learned that Microsoft Office has become a de facto means of sharing information between countries or agencies, and that the lack of selective access control within results in people *simply not sharing*. This major usability shortcoming impacts DoD coalition operations. This problem is not restricted to the defense setting: analogous problems can easily be identified in commercial settings. Additionally, both military computers and company-owned computers are often subject to strict conditions or restrictions regarding the softwares that can be installed and run.

Motivated by this state of affairs, this paper addresses the latter usability problem head-on, by showing that fine-grained access control solutions can be made available and naturally used in widely-deployed products. This paper introduces a Microsoft Excel add-in that enables cell encryption according to policies. The add-in is a single-page web application written in JavaScript that interacts with the object models in Excel using the JavaScript API for Office [1]. A minimal locally hosted (dockerized) backend service<sup>3</sup>, accessible through a REST API, enables to encrypt and decrypt using a CP-ABE scheme. After encryption, the document remains a *valid Excel document* and can be opened and read (without the encrypted cells) as *any* other `xlsx` document by *any* software. More precisely, the encrypted cells are stored in an XML custom part of the `xlsx` document through the JavaScript API. When our add-in is loaded from the Microsoft Excel software (or, e.g., from the Online Excel of Office 365) by a user in possession of a CP-ABE key, all the cells with a policy satisfiable by the key attributes will be decrypted and displayed. Henceforth, the same `xlsx` document can be shared with a wide audience while enabling selective access control at a cell level.

*Organization:* The outline of the paper is as follows. Section 2 introduces some necessary background on Microsoft Office add-ins and CP-ABE. Section 3 presents our Excel add-in, and a performance evaluation is presented in Section 4. Section 5 discusses potential alternative options for short-term adoption of similar selective sharing functionality based on standardized encryptions schemes but with some limitations. Finally, we conclude the paper and discuss future work in section 6.

## 2 Preliminaries

This section provide some common background and notation on Office add-ins, ciphertext-policy attribute-based encryption, and the Charm framework.

### 2.1 Microsoft Office Add-In

An Office add-in is a web application that is loaded from a browser inside of an Office application (desktop and online). An add-in is not installed on the host, but has its implementation hosted on a web server. Add-ins can be added in an Office application either by providing a XML manifest file (with the URL of the web application), or through the Office store.

<sup>3</sup> We note that such a backend service is just to simplify the development an experimentation, but in a production development,, the entire encryption and decryption could/should be performed on the client-side in the add-in webpage itself via JavaScript. We verified that this is possible and implemented it in later versions while writing this paper.

As with any web services, add-ins can access any web-based resources. Accessing and modifying information in the Office document is made possible by referencing the `office.js` file containing the JavaScript API for Office [1]. The add-in logic is developed in JavaScript:

```
Office.initialize = function () {
  // Office is ready
  $(document).ready(function () {
    // Implementation of add-in logic
  });
};
```

Once initialized, an add-in can access data in the underlying application. For example, the code in Fig. 1 recovers the content (the formulas) of the current selected cells in Excel.

```
Excel.run(function(ctx) {
  var range = ctx.workbook.getSelectedRange();
  range.load('formulas');
  return ctx.sync().then(function() {
    var content = range.formulas;
    // Process the content
  });
});
```

**Fig. 1.** Snippet of code to recover the content of a range of cells.

We refer to the official documentation for further detail on the Office add-in platform [3].

## 2.2 Access Structures

An access structure specifies the set of attributes required to gain access to some secrets.

**Definition 2.1 (Access structure).** Let  $\mathcal{U}$  be a universe of attributes. An access structure  $\mathbb{A}$  is a collection of non-empty subsets of  $\mathcal{U}$ . An access structure  $\mathbb{A}$  is called *monotone* if, for every  $B \subseteq C \subseteq \mathcal{U}$ ,  $B \in \mathbb{A} \Rightarrow C \in \mathbb{A}$ .

In this paper, we will define access control in terms of *policies* over attributes with AND and OR gates (cf. Section 3.6), that are then converted into access structures to be used by the CP-ABE scheme.

## 2.3 Ciphertext-policy ABE

Let  $\lambda$  denote the target bit-security of the cryptographic scheme (a.k.a, the security parameter). A ciphertext-policy ABE scheme CP-ABE = (Setup, Enc, KeyGen, Dec) is a tuple of probabilistic algorithms together with a message space  $\mathcal{M}$  that behave as follows:

- Setup takes as input the security parameter  $\lambda$  and outputs a public key  $pk$  and a master secret key  $msk$ .
- Enc takes as input the public key  $pk$ , a message  $m$  and an access structure  $\mathbb{A}$ , and outputs a ciphertext  $c$ .
- KeyGen takes as input the master secret key  $msk$  and a set of attributes  $S$ , and outputs a secret key  $sk$ .
- Dec takes as input the public key  $pk$ , a ciphertext  $c$  and a secret key  $sk$ , and outputs  $m^*$  or  $\perp$ .

A CP-ABE scheme must satisfy the following *correctness* condition: for all  $m \in \mathcal{M}$ , access structure  $\mathbb{A}$ , and set of attributes  $S \in \mathbb{A}$ , it holds that

$$\Pr \left[ \text{Dec}(pk, c, sk) \neq m \mid \begin{array}{l} (pk, msk) \leftarrow \text{Setup}(1^\lambda) \\ c \leftarrow \text{Enc}(pk, \mathbb{A}, m) \\ sk \leftarrow \text{KeyGen}(msk, S) \end{array} \right] \leq \text{negl}(\lambda),$$

where  $a \leftarrow A(b)$  denotes the output of the algorithm  $A$  when run on input  $b$  and  $\text{negl}(\lambda)$  denotes a negligible function, i.e., a function which is smaller than the inverse of any polynomial for large enough values of  $\lambda$ .

In this paper, we only consider fully-secure CP-ABE schemes. We recall here the intuition, and refer to [9, Sec. 2.3] for a formal definition. A CP-ABE scheme is fully-secure against chosen plaintext attacks if, at any time after the deployment of the ABE scheme, no group of colluding users can distinguish between encryption of two messages of their choice, under an access structure (a.k.a., a policy) of their choice, as long as no member of the group can decrypt on their own.

## 2.4 Charm

Charm [10] is a framework for prototyping advanced cryptosystems based on Python. It uses an hybrid design: performance intensive mathematical operations are implemented in native C modules, while cryptosystems themselves are written in Python. In particular, Charm uses the Pairing-Based Cryptography Library [4] for elliptic-curve generation, operations, and cryptographic pairing implementations.

The scheme we use in our system is FAME, a CP-ABE scheme proposed at CCS'2017 by Agrawal and Chase [9]. We use without modification the authors' implementation of FAME (as incorporated into Charm).

## 3 The Excel Add-in

This section presents our Excel add-in and its workflows.

### 3.1 Setting

Our add-in assumes the existence of the following entities/services:

- An entity  $O$ , who will create CP-ABE parameters

$$(\text{pk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda).$$

$O$  is the only entity knowing the master secret key  $\text{msk}$ , hence the only party that will be able to create secret keys  $\text{sk}$ 's associated to sets of attributes.

- A web service  $W$ , which will be hosting the add-in web application (cf. Section 2.1). This service may *or may not* be controlled by  $O$ .
- A service  $E$  (e.g., a web service accessible through a REST API), which will enable to encrypt with the CP-ABE scheme. This service may be completely independent of  $O$  and  $W$ ; it only implements the Enc operation of the CP-ABE scheme.
- A service  $D$  (e.g., a web service accessible through a REST API), which will enable to decrypt with the CP-ABE scheme. This service may be completely independent of  $O$ ,  $W$  and  $E$ ; it only implements the Dec operation of the CP-ABE scheme.
- $n$  entities  $P_i$ 's that will be issued secret keys by  $O$  over time; these will receive a protected spreadsheet and use the add-in to recover the information they are entitled to see.

Our add-in enables everyone (i.e., the above entities *or anybody else*) to create selectively protected spreadsheets. In particular, it will enable to select cells and encrypt them according to policies. Therefore, our add-in only requires to know which universe of attributes it should use to allow policy creation.

Additionally, in order to run the Enc and Dec algorithms, the services  $E$  and  $D$  need to know the public key  $\text{pk}$ . In our evaluation (Section 4), we assume the public key to be embedded in the services  $E$  and  $D$ . An alternative option could be for the add-in to be configured at load time with the public key and to send this public key along with the message and policy (resp., with the ciphertext and the secret key) every time it asks for encryption (resp., decryption).

### 3.2 (Offline) Key Distribution

This key distribution is *decoupled* from the Excel add-in, and may be performed offline and out-of-band.

The entity  $O$  is the only entity that can issue secret keys  $sk$ 's. In the following, we assume that  $O$  issued and shared one or more secret keys  $sk_{ij}$ 's to each party  $P_i$ . Note that a key  $sk$  will be used to decrypt *several spreadsheets over time* (as long as its attributes can decrypt cell policies).

In the following, we assume the  $sk_{ij}$ 's are stored in (say) JSON files.

### 3.3 Spreadsheet Creation

As recalled above, anyone using the add-in may activate the privacy protection in an Excel spreadsheet.

The activation of the add-in follows the following workflow:

1. Display a page to enable the activation of the privacy protection. This page asks to create an administrative password (Fig. 2).
2. Upon submission of a password  $p_a$ , apply the script password-based key derivation function thereon to obtain a key  $k_a$  (our add-in uses the `script-async` library [5]). (This administrative key will encrypt any administrative-related data.)
3. Store a salted hash  $H(k_a; n_a)$  of the key  $k_a$  in the

```
Office.context.document.settings
```

object. The hash is computed using the `TweetNaCl.js` library [6]. This object is saved in the Excel document, and will be accessible via any Excel application.

4. In order to enable cell encryption, a configuration file containing the list of all attributes needs to be loaded in the add-in. An example of such file is provided in Figure 3.
5. Store the list of attributes encrypted under  $k_a$  in the settings.

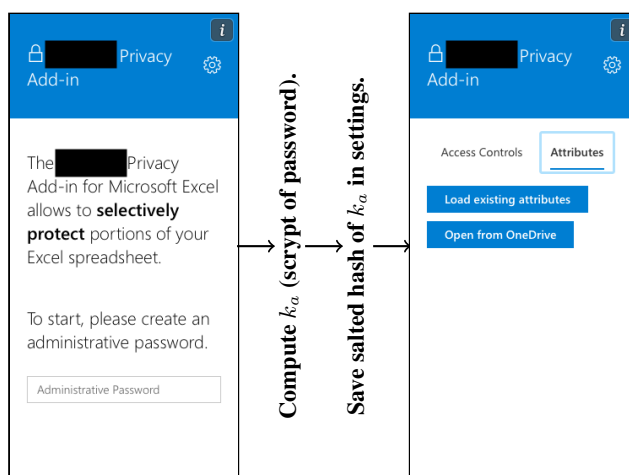


Fig. 2. Workflow to create a privacy-protected Excel spreadsheet. (The screenshots are anonymized for submission.)

At the end of the spreadsheet creation, the settings contain a hash  $(n_a, H(k_a; n_a))$  of the administrative key  $k_a$ , and the list of attributes encrypted under  $k_a$ .

### 3.4 Encryption

In this subsection, assume a user  $U$  wants to add cell encryption in a  $\times 1.5 \times$  document created as in Section 3.3. Figure 4 shows a screenshot of the add-in after encryption of three ranges of cells with two policies.

```

{
  "Continent": ["Africa", "Antarctica", "Asia", "Australia", "Europe", "North America", "South
    America"],
  "Country": ["Canada", "China", "France", "Japan", "Russia", "UK"],
  "Trust level": [1, 2, 3, 4, 5],
  "Organization": ["BRICS", "G20", "G7", "NATO", "WTO"],
}

```

**Fig. 3.** An example of configuration file containing the list of all attributes.

**Authentication** In the current version of the add-in, we only enable cell encryption when  $U$  knows the administrative password. This is not necessary and one may choose to remove this authentication step. Note that from Section 3.3, only the attributes are stored in the settings; future work may include additional (encrypted) content in the settings, which explains why we implemented this more general approach.

The workflow at load time is as follows:

1. Check the presence of a hash ( $n_a, H(k_a; n_a)$ ) in the settings. If defined, display a login screen.
2. Upon submission of a password  $p_u$  by  $U$ , apply the script password-based key derivation function thereon to obtain a key  $k_u$  (our add-in uses the `script-async` library). If  $H(k_u; n_a) = H(k_a; n_a)$ , decrypt the attributes from the settings (if any) and populate the UI accordingly; if not, go back to Step 1.

Upon success of Step 2,  $U$  will be considered “authenticated”.

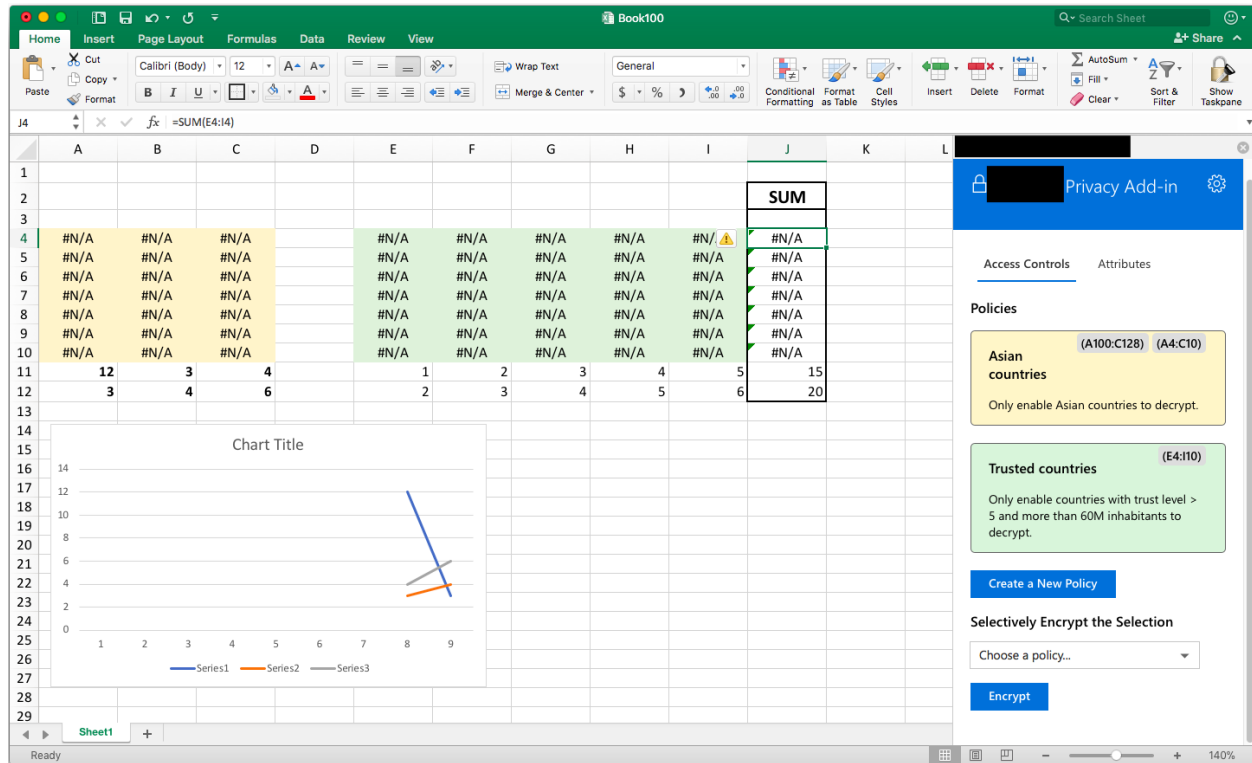
**Encryption** Assume  $U$  wants to encrypt a range of cells (say A1:B4) under a policy of her choice.  $U$  will select the cells in the range (A1:B4) in Excel, will use the policy UI to create a policy (cf. Fig. 5), and will click on the “Encrypt” button.

Upon click, the encryption workflow is as follows:

1. Get the content of the selected range (see Fig. 1 for an excerpt of our JavaScript code); without loss of generality, we assume the content is a  $n \times m$  matrix of strings  $C$ ;<sup>4</sup>
2. Generate a random key  $k_c$  (using `TweetNaCl.js`);
3. Encrypt every string  $C[i][j]$  with the secret key  $k_c$  and obtain  $E[i][j]$  (using the `TweetNaCl.js` symmetric encryption scheme);
4. Recover the policy as a string  $P$  from the UI;
5. Use the service  $E$  to encrypt  $k_c$  under  $P$  and obtain a ciphertext  $c$ ;
6. Store  $(c, \{E[i][j]\}_{i,j})$  in a custom XML part object in the document using the JavaScript API for Office.
7. Clear the content of the cells; e.g., our add-in replaces each of the  $C[i][j]$  by #N/A. We made this choice because each formula including an encrypted cell will then automatically display #N/A (cf. Fig. 4).

Note that our add-in uses hybrid encryption (i.e., data encapsulation using symmetric encryption and a public key encryption of the symmetric key), that is instead of encrypting each  $C[i][j]$  using the CP-ABE encryption scheme, it generates a symmetric key  $k_c$ , encrypt all the cells under  $k_c$  using a symmetric encryption scheme, and encrypts  $k_c$  under the CP-ABE scheme (with the public parameters). The reason is threefold: (1) encrypting/decrypting under a symmetric encryption scheme is much faster than encrypting/decrypting with the ABE scheme; (2) this enables to perform cell encryption locally rather than sending the cell content to the external service  $E$ ; and (3) when encrypting with the ABE scheme, the ciphertext is significantly larger than the message (by several order of magnitudes), whereas it remains of roughly the same size when using the symmetric encryption scheme. As such, as soon as we encrypt two cells with the hybrid method, we are more efficient in time and space than encrypting both cells with the ABE scheme. We provide concrete numbers in Table 2.

<sup>4</sup> Note that in our add-in, we load the *formulas* of the cells, and not the displayed text values (Fig. 1). This enables to recover cell inputs, such as “=SUM(A1:A10)”, that compute over cell ranges, and hence to keep the *dynamicity* of the spreadsheet.



**Fig. 4.** Screenshot of the add-in after encrypting three ranges of cells: A4:C10 and A100:C128 are encrypted with a policy (continent == Asia) (named “Asian countries” by the user), and E4:I10 is encrypted with a policy (trust > 5) OR (population >= 60000000) (named “Trusted countries” by the user). Column J computes the sum of the values in the columns E to I for each row; note that it outputs #N/A when the cells are encrypted. The chart display the cells in the range A4:C12; note that only unencrypted values are visible in the chart.

As a side remark, note that most implementations using public-key cryptography today use a hybrid system. Examples include the TLS protocol, which uses a public-key mechanism for key exchange (such as Diffie–Hellman) and a symmetric-key mechanism for data encapsulation (such as AES), OpenPGP and PKCS #7 (see discussion about alternative approaches for achieving a subset of the functionality but would be viable in the short-term in section 5).

### 3.5 Decryption

In this subsection, assume a user  $P_u$ , who has been issued one or more secret keys  $sk_{uv}$ ’s for attribute sets  $S_{uv}$ ’s by  $O$ , gets access to a spreadsheet with several encrypted cells as in Section 3.4. At load time, the add-in checks the presence of encrypted cells; if present, it displays a screen to drag and drop secret keys.

Upon drag of a key file corresponding to a CP-ABE secret key  $sk \in \{sk_{uv}\}_v$ , the decryption workflow is as follows:

1. For every group of encrypted cells as generated by Section 3.4, recover  $(c_\ell, \{E_\ell[i][j]\}_{i,j})$  from the custom XML part object.
2. For every  $c_\ell$ , use the service  $D$  to decrypt  $c_\ell$  with  $sk$ , and obtain  $m_\ell$  or  $\perp$ .
3. When it decrypts correctly, define  $k_c = m_\ell$  and decrypt the cells  $E_\ell[i][j]$  to recover  $C_\ell[i][j]$ .
4. Replace the content of the cell range by  $C_\ell[i][j]$ .

The CP-ABE scheme ensures that, if the attributes embedded in  $sk$  do not satisfy the policy associated to the ciphertext  $c_\ell$ ,  $P_i$  cannot recover the corresponding symmetric key. The symmetric encryption scheme ensures that the

**Fig. 5.** Pop-up that enables creation of conjunctive normal forms policies, that is policies of the form  $(\text{expr}11 \text{ OR } \dots \text{ OR } \text{expr}1i) \text{ AND } (\text{expr}21 \text{ OR } \dots \text{ OR } \text{expr}2j) \text{ AND } (\text{expr}31 \text{ OR } \dots \text{ OR } \text{expr}3k)$ .

content of the cells remains secret to anyone that would not know the symmetric key. An important benefit of the hybrid approach is that the service  $D$  never gets to know the *content* of the cells either; instead decryption is done locally within the application itself.

Finally, note that the ABE scheme is secure against collusions. For example, assume a cell is encrypted under the policy

$$(\text{continent} == \text{Asia}) \text{ AND } (\text{continent} == \text{Europe}).$$

Even if participant  $P_1$  (resp.  $P_2$ ) has the attribute  $\text{continent} == \text{Asia}$  (resp.  $\text{continent} == \text{Europe}$ ),  $P_1$  and  $P_2$  together cannot combine their key to decrypt the ciphertext associated to the cell encryption, and therefore do not learn the cleartext content the cell.<sup>5</sup>

### 3.6 Expressiveness of Policies

To increase usability of our add-in, we developed a policy creation UI (Fig. 5) that allows a user to easily create policies, eventually expressible as Boolean expressions<sup>6</sup> with operators AND and OR of predicates of the form

$$\text{name} == \text{value}$$

for string values, and

$$\begin{aligned} &\text{name} == \text{value} \\ &\text{name} >= \text{value} \\ &\text{name} > \text{value} \\ &\text{name} <= \text{value} \\ &\text{name} < \text{value} \end{aligned} \tag{1}$$

for numerical values.

For example, this allows the creation of policies of the form:

<sup>5</sup> Note that this policy makes sense; e.g., Russia or Turkey could be potential intended recipients of such a policy.

<sup>6</sup> More precisely, it allows the creation of conjunctive normal forms (CNF).



```
((continent == Europe) OR (trust >= 3)) AND (org == NATO) AND (key_valid_until
> 1518523199),
```

to share data with a trusted country or a European country, part of the NATO organization, with a valid key. Indeed, the last predicate of the above policy allows for key revocation by including a numerical attribute `key_value_until` in the keys, as proposed in [13, Sec. 4.3].

**Attributes in the Keys** Recall that at key generation time, CP-ABE schemes take as input a set of attributes  $S$ . In our add-in, attributes in the keys are specified by name/value:

name = value.

When `value` is a string, we add to the set  $S$  the string `"name:value"`. When `value` is a  $k$ -bit number, we use a simple trick (already mentioned in [13, Sec. 4.3]) that decomposes the number into its bits, adding the  $k$  (string) attributes to the set  $S$ :

```
"name:v_{k-1}*****"
      :
      :
"name:*****v_1*"
"name:*****v_0"
```

where  $\text{value} = \sum_{i=0}^k v_i \cdot 2^i, v_i \in \{0, 1\}$ .

**Policies in Ciphertexts** Recall that at encryption time, CP-ABE schemes take as input access structures rather than a policy string; we therefore use the Charm policy parser [10] to convert our policies. Unfortunately, while the current policy parser of Charm explicitly parses<sup>7</sup> the predicates for numerical values of Eq. (1), any such predicate is replaced by the string `name` and disregards the value altogether (see the culprit function<sup>8</sup> on Fig. 6).

```
# convert 'attr < value' to a binary tree based on 'or' and 'and'
def parseNumConditional(s, loc, toks):
    print("print: %s" % toks)
    return BinNode(toks[0])
```

**Fig. 6.** Extract from the `charm/charm/toolbox/policytree.py` file in Charm that does not handle correctly numerical predicates. `toks` is a list containing three strings: the name, the operator, and the value.

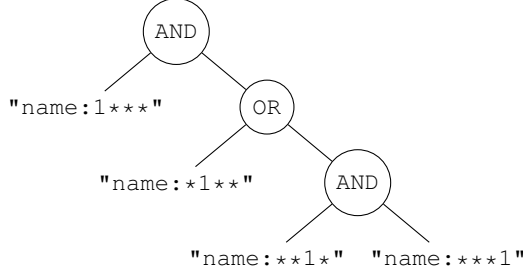
In our add-in, we modified the Charm policy parser to handle the predicates of Eq. (1). Using again the bit decomposition of  $\text{value} = \sum_{i=0}^k v_i \cdot 2^i, v_i \in \{0, 1\}$ , we use a simple tree implementing the operator (see Fig. 7 or [13, Fig. 1]) using the AND and OR operators.

**Number of Bits** An important shortcoming of the approach described in Section 3.6 is that one has to be careful with the expected length of the numerical values. Indeed, assume that `name = 16`; the transformation of Section 3.6 yields that the key attributes set contains

```
name:1*****
name:*0****
name:**0**
name:***0*
name:****0
```

<sup>7</sup> <https://github.com/JHUISI/charm/blob/dev/charm/toolbox/policytree.py#L52>

<sup>8</sup> <https://github.com/JHUISI/charm/blob/dev/charm/toolbox/policytree.py#L20>



**Fig. 7.** Tree implementing the attribute  $\text{name} \geq 11$ . The Boolean expression derived from the tree evaluates to true when the key contains either (a) "name:1\*\*\*" and "name:\*1\*\*"; or (b) "name:1\*\*\*", "name:\*\*1\*" and "name:\*\*\*1"; case (a) captures  $\text{name} \geq 12$  and case (b) captures  $\text{name} \in \{11, 15\}$ .

The key would therefore not decrypt a ciphertext encrypted under the policy of Fig. 7 (while it should).

In our implementation, we enable specifying the number of bits of numerical attributes, defaulting to 32-bit numbers for usability. An important caveat of defaulting to 32 bits is each tree policies may contain up to 32 attributes, which impacts the performance of the online encryption with the CP-ABE scheme (see Table 1).

Number of bits of the numerical values of the attributes	4	8	12	16	20	24	28	32
KeyGen (attributes: $a = k$ )	31 ms	54 ms	77 ms	100 ms	123 ms	146 ms	168 ms	191 ms
Enc (policy: $a == k$ )	27 ms	50 ms	74 ms	97 ms	121 ms	144 ms	168 ms	191 ms
Dec	26 ms	26 ms	26 ms	27 ms	27 ms	27 ms	27 ms	27 ms
Enc (policy: $a \leq n$ )	28 ms	51 ms	75 ms	98 ms	119 ms	142 ms	165 ms	186 ms
Dec	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms
Enc (policy: $a < m$ )	23 ms	45 ms	70 ms	82 ms	99 ms	134 ms	160 ms	184 ms
Dec	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms	26 ms

**Table 1.** Average performances of the KeyGen, Enc, and Dec operations where the ciphertext is associated to a policy  $a == k$  (resp.,  $a \leq n$ , resp.  $a < m$ ) and the key is associated to an attribute  $a = k$ , for  $N$ -bit integers  $k, n, m$  and  $k \leq n$  and  $k < m$ . The CP-ABE scheme is FAME instantiated in the Charm framework on a Intel Pentium CPU G4400 at 3.30GHz.

## 4 Evaluation and Performances

**Choice of CP-ABE.** As mentioned in the introduction, efficient and unbounded CP-ABE schemes based on well-established security assumptions have been proposed recently. In our add-in, we use the FAME CP-ABE scheme over the MNT224 curve introduced at CCS'2017 by Agrawal and Chase [9]. As far as we know, FAME is the most efficient CP-ABE scheme today (at the time of developing the add-ins and writing of this paper) for the encryption and decryption operations [9, Sec. 5].

**Docker-Compose.** Our test environment runs three Docker containers: a `nginx:latest` container that serves the add-in web page (the web service  $W$ ), a `python:latest` container accessible through a REST API to access the services  $E$  and  $D$ , and finally a `nginx:latest` proxy container that listens on port 443 and redirects either to the add-in or to the backend. The Python container uses the FAME implementation of the Charm framework [10] for CP-ABE encryption and decryption.

**Environment.** The host is a MacBook Air (Late 2014) running macOS High Sierra 10.13.3 with a 1.7 GHz Intel Core i7. The version of the Docker engine is 17.12.0-ce and the version of Excel is 16.9 (180116).

# of cells	1	10	100	1,000	10,000
Encryption with policy <b>P-I</b>	150 ms	150 ms	163 ms	181 ms	580 ms
Encryption with policy <b>P-II</b>	397 ms	407 ms	416 ms	421 ms	837 ms
Encryption with policy <b>P-III</b>	386 ms	394 ms	410 ms	418 ms	848 ms
Encryption with policy <b>P-IV</b>	1,410 ms	1,417 ms	1,423 ms	1,434 ms	1,614 ms

**Table 2.** Benchmark of the encryption workflow (Section 3.4) according to different policies, on 1 to 10,000 cells.

**Easy-to-use.** Our add-in is very easy to use; it only requires a user to install the add-in (e.g., via the integrated add-in store) and to specify the attributes that will be used to construct the encryption policies (e.g., using a configuration file). In particular, it *does not modify Excel in any way* and *does not require additional software* to be installed on the machine.

## 4.1 Encryption

**Setting.** We start from  $5 \times 1s \times$  documents, containing respectively 1, 10, 100, 1 000, and 10 000 cells with value #N/A. We report the time to encrypt those cells against four policies (see below), and the size of the resulting documents. Note that our baseline documents contains #N/A as text because, in Step 7 of our encryption workflow, we clear the cells by replacing their content by #N/A: keeping the same content displayed in the cells enables us to measure as accurately as possible the size overhead due to the encryption.<sup>9</sup>

**Policies.** We measure the performances of our encryption workflow with four policies.

**P-I:** `(name == value);`

The first policy is a simple policy that checks the presence of *one* attribute `name = value`, where `value` is a string, in the secret key. This is the simplest policy that can be defined.

**P-II:** `(name == n)` where  $n$  is a 32-bit number;

The second policy is a policy that checks that the key has been created for `(name = n)`. Recall from Section 3.6 that the key will contain 32 attributes of the form `name:***b***` where  $b \in \{0, 1\}$  and a varying number of `*`. The policy checks equality, i.e., checks that the key contains all the aforementioned attributes.

**P-III:** `((name1 == value1) OR (name2 == value2)) AND (name3 > n)` where  $n$  is a 32-bit number;

The third policy has the form of a policy created by our UI (Fig. 5). To enable decryption, a key needs to contain at least 33 attributes (the 32 attributes for the numeral values, and at least one attribute of `name1 == value1` or `name2 == value2`). Recall from Section 3.6 that the policy is a tree with up to 33 leaves.

**P-IV:** `((name1 == n1) OR (name2 == n2) OR (name3 == n3) OR (name4 == n4)) AND (name5 > n5)`, where  $n_i, i \in \{1, \dots, 5\}$  are 32-bit numbers;

The fourth policy is a “bad” policy, in the sense that it yields a Boolean formula with up to  $5 \times 32 = 160$  predicates. As we will see, this yields a large ciphertext and impacts the encryption time.

**Timings.** Table 2 reports benchmarks for the encryption workflow (Section 3.4), that is the time it takes from the moment a user clicks ‘Encrypt’ and the moment the content of the cells is cleared (Step 7).

These timings illustrate the interest of hybrid encryption (Section 3.4): encrypting 1 or 1 000 cells takes approximately the same time. These timings also show that such an add-in is usable: encrypting 10,000 cells with a complex policy (i.e., that involves a lot of attributes) takes about 1.5s when using Python in a Docker container on a standard laptop. Significant gains are to be expected by running an efficient implementation of the CP-ABE scheme natively on a server.

**Size.** Figure 8 reports the sizes of the  $1s \times$  documents after encrypting 1 to 10 000 cells according to the above policies. (Note that the  $x$  axis is logarithmic.)

<sup>9</sup> Obviously, the longer the text in the cells, the larger the documents will be. We use the default secret-key authenticated encryption of TweetNaCl.js (XSalsa20-Poly1305); hence the size of each ciphertext is 16 bytes longer than the original message.

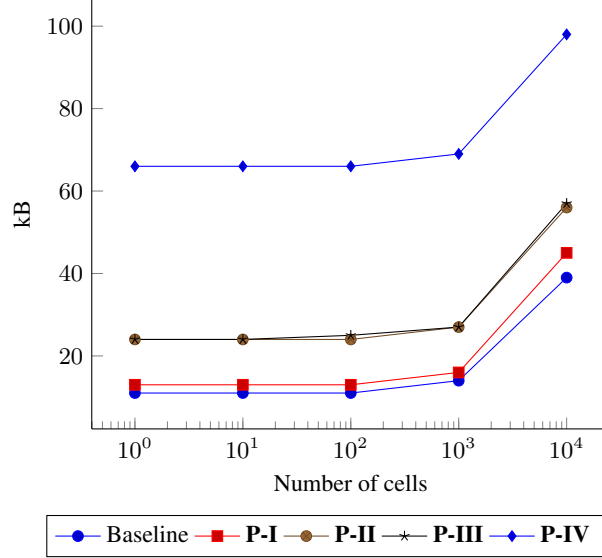


Fig. 8. Plot of the size of the  $x_{1s}x$  documents after encrypting 1 to 10,000 cells according to policies **P-I** to **P-IV**.

This figure shows that, as expected from the workflow of Section 3.4, there is a one-time size increase corresponding to the encryption of the key under the CP-ABE scheme (difference at the leftmost of the plot between the baseline size and the sizes after encrypting one cell), and then a small overhead corresponding to the encryption of the cells. This overhead grows linearly with the number of cells encrypted. It follows that encrypting 10,000 cells according to the complex policy **P-IV** only increases the document size by about 60kB.

## 4.2 Decryption

As shown on Table 1, regardless of the policy, the decryption time is very efficient. Indeed, decrypting requires to compute 6 cryptographic *pairings* (bilinear maps) over elliptic curves, 6 multiplications in the target group, and  $6I + 3$  multiplications in the input group, where  $I$  is the number of attributes used in decryption. Since multiplying in the input group is three order of magnitude faster than computing a pairing (cf. [9, Table 5.1]), the decryption time is nearly independent of the number of attributes involved. Therefore, the execution time of the decryption workflow (Section 3.5) amounts to the asynchronous execution of the JavaScript in the browser within the Excel software (plus network communication). In Table 3 we report average time (over 10 runs) to decrypt 100 to 1,000 cells, encrypted as 10 sets of 100 cells according to random policies of the form **P-I**, **P-II**, **P-III**, and **P-IV**. These timings show that our (unoptimized) implementation already achieves good performance.

Number of cells that can be decrypted	Average time
1 · 100	1,082 ms
3 · 100	1,295 ms
6 · 100	1,668 ms
10 · 100	1,851 ms

Table 3. Benchmark of the decryption workflow (Section 3.5) on 10 sets of 100 encrypted cells according to a random policy of the form **P-I**, **P-II**, **P-III**, and **P-IV**.

## 5 Short-term Adoption: Policy-based Encryption without Collusion-Resistance via Multi-key Hybrid Encryption (Using Standardized Schemes)

The deployment of ABE in production systems, e.g., in government and commercial applications, remains limited. Currently, to the best of our knowledge, no widely deployed commercial authoring software platforms and products use ABE. The root cause of this (in the USA) may be because ABE has not been standardized yet by well known standardization bodies that develop, endorse, and maintain such national and international standards, e.g., the National Institute for Standards and Technology (NIST) in the USA. While ABE has not (yet) been standardized in the USA, there are recent efforts in that direction by the European Telecommunications Standards Institute (ETSI)<sup>10</sup>.

Developing new cryptographic standards is a process that takes several years (as it should) due to its complexity and importance as illustrated by the ongoing<sup>11</sup> NIST effort to standardize post-quantum cryptography (PQC). While we acknowledge that standardizing PQC is a much larger and challenging effort compared to standardizing ABE, nevertheless, we do not expect any long-term standard to be initiated, completed, and then ratified in the next two to three years, especially if one considers a timelines similar to standardizing PQC.

A natural question then becomes “*is there a way to only utilize standard public-key/asymmetric and symmetric schemes and emulate most of the functionality and guarantees provided by ABE in some settings?*”. We sketch here a potential approach that we argue works in many enterprise settings. We stress that this is an informal treatment to argue that short-term secure selective sharing solutions may be designed and deployed, building upon the in-app cryptographically-enforced framework developed in this paper, until ABE is standardized and ready for commercial wide-scale adoption. Specifically, we focus on settings where one is not concerned about a built-in technical solution to collusion-resistance from users and insiders in the enterprise. For example, if the policy is encrypting to multiple parties, where each party by itself should be able to decrypt (i.e., an OR clause), then there is no potential (nor reason) for collusion between parties. There are a lot of settings and application where an encrypted object should be restricted to a group of employees in the enterprise, and each of them alone can, and should be able to, decrypt.

**Representing Encryption Policies in Disjunctive Normal Form (DNF).** While in the ABE case, policies were expressed in Conjunctive Normal Form (CNF) form (see Section 3.6), one can easily convert a policy into a DNF form. Whether CNF or DNF representations is preferable will depend on the application. Some functions can be succinctly represented in DNF whereas others are represented more succinctly in CNF; switching between these representations can involve an exponential increase in size [22]. We outline below techniques to use (standardized) public-key/asymmetric encryption schemes in a blackbox manner to realize AND and OR clauses. It will be up to the application to decide how to combine these into encryptions that represent DNF or CNF. It is important to stress that this encryption is only used to wrap a random short symmetric key (e.g., an AES key) as typically used in hybrid encryption.

**Encrypting to OR Clauses.** The approach to encrypt an OR clause is to encrypt the symmetric key  $k$  used to encrypt the data object ( $m$ ) with different public-keys, where each public-key corresponds to an attribute in the clause. For example, if the clause is  $a_1$  OR  $a_2$  OR  $a_3$ , where  $a_i$  corresponds to  $pk_i$ , then an encryption of data  $m$  and symmetric key  $k$  for such a clause would be  $\{E_{pk_1}^{a_1}(k)||E_{pk_2}^{a_2}(k)||E_{pk_3}^{a_3}(k)||E_k^s(m)\}$ , where  $||$  denotes concatenation and  $E_{pk_i}^{a_i}(\cdot)$  denotes public-key/asymmetric encryption with key  $pk_i$  for attribute  $a_i$ , and  $E_k^s(\cdot)$  denotes symmetric key encryption with key  $s$ .

**Encrypting to AND Clauses.** There are two typical approaches to perform an encryption of an AND clause.

The first approach uses nested re-encryption, it performs sequential re-encryption of the symmetric key  $k$  and ciphertexts resulting from encrypting it under the different public-keys corresponding attributes in the AND clause. For example, if the AND clause is  $a_1$  AND  $a_2$  AND  $a_3$ , where  $a_i$  corresponds to  $pk_i$ , then encryption of data  $m$  with symmetric key  $k$  for such a clause would be  $\{E_{pk_3}^{a_3}(E_{pk_2}^{a_2}(E_{pk_1}^{a_1}(k)))||E_k^s(m)\}$ .

The second approach is to use additive (or another forms if t-out-of-n decryption is required) secret sharing of the symmetric key  $k$  to be encrypted, and then encrypt each share under different public keys. For example, if the AND clause is  $a_1$  AND  $a_2$  AND  $a_3$ , where  $a_i$  corresponds to  $pk_i$ , then encryption of data  $m$  with symmetric key  $k$  for such a clause would be  $\{E_{pk_1}^a([k]_1)||E_{pk_2}^a([k]_2)||E_{pk_3}^a([k]_3)||E_k^s(m)\}$ , where  $[k]_i$  is share  $i$  of the key  $k$ .

<sup>10</sup> <https://www.etsi.org/newsroom/press-releases/1328-2018-08-press-etsi-releases-cryptographic-standards-for-secure-access-control>

<sup>11</sup> <https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline>

The trade-off offered by the two approaches above is easy to see: the first approach requires less space but encryption and decryption cannot be parallelized, while in the second approach encryption and decryption can be parallelized, but would require more space.

**Security.** Given that the actual data is encrypted using a standard symmetric authenticated encryption scheme (e.g., the AES-GCM authenticated encryption scheme) with a random key  $k$ , the data confidentiality is ensured when  $k$  remains secret. We argue below security of the key encapsulation mechanism (KEM) used to encrypt  $k$  for both an AND clause and an OR clause.

*Security of an AND clause:* The key  $k$  can be secret shared into  $l$  shares depending on the number of  $l$  literals/attributes in the AND clause. Due to the properties of secret sharing, each share of  $k$  ( $[k]_i$ ) by itself will be a random string. Each  $[k]_i$  will then be encrypted independently via the (asymmetric) public-key encryption scheme ( $E_{pk_i}^{a_i}(\cdot)$ ) and a different public-key  $pk_i$ . It is easy to argue by contradiction that, if such a construction is insecure, then a single application of the underlying  $E_{pk_i}^{a_i}(\cdot)$  is insecure because one could always concatenate a single such encryption with other encryptions of random messages for random public-keys and pass them to an adversary that breaks such a concatenation produced from an AND clause, thus resulting in a break of the underlying encryption scheme.

*Security of an OR clause:* We note that the encryption of an OR clause is essentially a multi-receiver KEM encrypting a random symmetric key used in a data encapsulation mechanism (DEM) approach. This is the approach utilized in encrypting email in well used protocols such as S/MIME<sup>12</sup>. A formal security treatment of this approach is outside the scope of this paper, but we report here informally the essence of why this approach is secure. If one can break the multi-receiver use of an appropriately chosen CCA-secure public-key encryption used as a KEM mechanism (with different public-keys), then one can devise a reduction from the multi-receiver KEM used above to a single receiver KEM and thus break the security of the underlying. The reduction would generate several ciphertexts of 0 and 1 and pair them with the two given challenge encryptions, and pass them to the multi-receiver KEM adversary to break the ones it could and then use this break to distinguish the two challenge encryptions.

**Performance Overhead.** We give below a high-level estimate of the encryption/decryption delay and computational overhead involved therein. We also assess the space overhead in the proposed approach.

*Computational Overhead and Delay from Decryption:* Assuming policies in DNF form with less than 10 OR clauses, each containing less than 10 attributes combined via an AND clause, one would have to do at most 100 public-key encryptions. As a rule of thumb, a typical public-key encryption is on the order of (or less than) one *msec* so such encryptions and decryptions will require less than a second. We note that while opening a large MS Office document is fast, it still is a bit perceptible to the user, i.e., not instant and may take a fraction of a second or even a full second. We argue that extending this by several hundred *msec* will be almost imperceptible to users. Finally, note that the encryptions and decryptions corresponding to the OR clauses are independent and can be easily performed in parallel. Encryptions and decryptions corresponding to AND clauses can also be parallelized if the secret sharing based technique described above is utilized.

*Increase in File Size:* The space overhead for the encryption of the actual data object is minimal as it is encrypted only once using a symmetric encryption scheme (e.g., AES) and a randomly generated key. The random symmetric key is then encrypted via public key several times to satisfy a policy that will at least contain two OR clauses, one for the originator of the encryption and one for the recipient of that encrypted data field. (We note though that it is likely that in enterprise settings, an additional OR clause may be added to the policy so that central IT (or similar organizations) can recover encrypted content belonging to the enterprise if employees thereof leave. This clause may be such that the symmetric key is secret shared and each share is encrypted with a different key belonging to different entities in the enterprise' IT or security departments.)

**Limitations:** One obvious limitation of this approach outlined above is that it only works for small policies, e.g., with a small number of clauses each with a few attributes. This approach also provides no collision resistance for AND clauses, but we argue that if each policy only has one AND clause corresponding to the recovery term described above, then it may be acceptable as if individuals high up and with significant privilege are acting malicious they could override policies and/or recover sensitive data through other means. The approach exhibits a linear overhead in the encryption size in the number of OR clauses and will require multiple public-key operations for encryption

<sup>12</sup> <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-49.pdf>

and decryption, but such computationally expensive operations can be easily parallelized when both encrypting and decrypting.

## 6 Conclusion and Future Work

This paper investigates and addresses a major usability hurdle: the lack of selective fine-grained access control in widely deployed enterprise products, and in particular in Microsoft Office products which are a de facto authoring means and often used to share information in government and private settings. More precisely, we present a user friendly way to achieve selective fine-grained protection of information in Excel by developing an Excel add-in, using the JavaScript API for Office. Our add-in brings the benefits of attribute-based encryption within spreadsheets. Using hybrid encryption, we show that it is possible to encrypt the cells' content locally, and minimize the size of the overhead due to encryption. Our add-in interacts with the state-of-the-art CP-ABE encryption scheme FAME (proposed at CCS'2017) and offers good performance and usability in our test environment.

*Future Work.* This paper is only the first step in a promising journey bringing ABE to widely used enterprise software products. An immediate next step will be to extend the current functionality to other products of the Office suite, such as PowerPoint. While similar add-ins and extensions to Word and Outlook have also been developed, the current API for PowerPoint seems more limited. This paper motivates extensions to the JavaScript API to enable fine-grained modifications in *all* Office applications. Furthermore, it is likely that a similar approach is possible to implement for Google Workplace/Suite applications (e.g., Google Docs, Google Spreadsheet), for which we have preliminary implementations proving the viability of developing similar add-ins. Contrary to the JavaScript API for Office, the Google add-ins framework only allows execution of server-side JavaScript code, which is a significant technical hurdle if data being encrypted (and keys) cannot (and should not) be exposed to cloud providers. We also envision developing add-ins for different web-browsers (e.g., to perform selective sharing/revealing of the content of a webpage and/or web-based applications) and add-ins for GMail. Future work could also investigate developing extension of the add-in(s) to support differential privacy [16].

## 7 Acknowledgments

The authors thank Tim Ellis, Ron Moore, and Karen Myers for helpful discussions and suggestions. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under Contract No. N66001-15-C-4071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or SSC Pacific. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

## References

1. Javascript API for Office. <https://dev.office.com/reference/add-ins/javascript-api-for-office>.
2. National Defense Authorization Act for the fiscal year 2000. <https://www.congress.gov/106/plaws/publ65/PLAW-106publ65.pdf>.
3. Office add-ins platform overview. <https://docs.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins>.
4. PBC library. <https://crypto.stanford.edu/pbc/>.
5. scrypt-async. <https://github.com/dchest/scrypt-async-js>.
6. TweetNaCl.js. <https://tweetnacl.js.org/>.
7. Using Excel services to share pieces and parts of Excel workbooks. <https://support.office.com/en-us/article/using-excel-services-to-share-pieces-and-parts-of-excel-workbooks-c9630a25-4c0a-43aa-8a93-510adb17b550>.
8. Zetroit LLC. <http://www.zetroit.com>.

9. S. Agrawal and M. Chase. FAME: Fast attribute-based message encryption. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 665–682. ACM Press, Oct. / Nov. 2017.
10. J. A. Akinyele, C. Garman, I. Miers, M. W. Pagano, M. Rushanan, M. Green, and A. D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013. <https://github.com/JHUISI/charm>.
11. N. Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 557–577. Springer, Heidelberg, May 2014.
12. N. Attrapadung. Dual system encryption framework in prime-order groups via computational pair encodings. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 591–623. Springer, Heidelberg, Dec. 2016.
13. J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society Press, May 2007.
14. J. Chen, R. Gay, and H. Wee. Improved dual system ABE in prime-order groups via predicate encodings. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 595–624. Springer, Heidelberg, Apr. 2015.
15. J. Chen, J. Gong, L. Kowalczyk, and H. Wee. Unbounded ABE via bilinear entropy expansion, revisited. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 503–534. Springer, Heidelberg, Apr. / May 2018.
16. C. Dwork. Differential privacy (invited paper). In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 1–12. Springer, Heidelberg, July 2006.
17. S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
18. R. Goyal, V. Koppula, and B. Waters. Semi-adaptive security and bundling functionalities made generic and easy. In M. Hirt and A. D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 361–388. Springer, Heidelberg, Oct. / Nov. 2016.
19. L. Kowalczyk and A. B. Lewko. Bilinear entropy expansion from the decisional linear assumption. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 524–541. Springer, Heidelberg, Aug. 2015.
20. A. B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 318–335. Springer, Heidelberg, Apr. 2012.
21. A. B. Lewko and B. Waters. Unbounded HIBE and attribute-based encryption. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 547–567. Springer, Heidelberg, May 2011.
22. P. B. Miltersen, J. Radhakrishnan, and I. Wegener. On converting cnf to dnf. *Theoretical Computer Science*, 347(1):325–335, 2005.
23. T. Okamoto and K. Takashima. Fully secure unbounded inner-product and attribute-based encryption. In X. Wang and K. Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 349–366. Springer, Heidelberg, Dec. 2012.
24. Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 463–474. ACM Press, Nov. 2013.
25. A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.