# Brief Survey of Cryptographic Attribute-Based Access Control Systems

Croix Gyurek
*Department of Mathematical Sciences*
*IUPUI*
Indianapolis, IN, USA
crgyurek@iu.edu

Kritika Verma
*Department of Engineering and Computer Science*
*Syracuse University*
Syracuse, NY, USA
krverma@syr.edu

***Abstract*—With the growing number of attackers that are trying to access important data files in big corporate companies or even small but major companies, technology also needs to move with it. Access control allows us to control who can and cannot view certain documents. One method for enforcing this is Attribute-Based Access Control (ABAC), where files are allowed or blocked based on "attributes", which are binary properties that a user can have or not have. Users can have multiple attributes and policies can be based on arbitrary Boolean functions of these attributes, so implementations of ABAC must be designed specifically for these policies. In this research paper, we surveyed three main research papers that used different schemes to design a stronger and more efficient ABAC model. All three methods relied on bilinear maps to allow the decryption step to derive mask values from the encryption step based on the user's key, without allowing multi-user collusion or key forgery. Two out of the three papers used the Linear Secret Sharing Scheme (LSSS) matrix to represent policies numerically, and one of them used recursion and polynomial interpolation in their decryption step. All three systems claim to be resistant to attacks, but only two provided explicit hardness assumptions to justify such claims, and one paper relied on the hardness of four different, though related, problems to maintain all security. In this paper, we take a look at all three papers and compare the three papers with each other. We also discuss the efficiency of the schemes as the number of attributes increase.**

***Index Terms*—access control, cryptography, attribute-based access control, bilinear maps, Linear Secret Sharing Scheme, survey**

## I. Introduction

The majority of security systems in today's world are set to protect data from external attacks. Access control controls what the user can and cannot access [2]. This includes specific files that everyone in a group should not be able to access. There are multiple policies that can be used that depend on the characteristic of the environment that needs to be protected. In this research paper, we focus on different attribute-based access control systems [1] which is a type of access control that is limited by computational languages and the richness of attributes that are available. This makes attribute-based access control suitable for environments that are changing rapidly.

In this research paper, we have read three papers [4]- [6] that are based on attribute-based access control but use different methods for encrypting and decrypting. We have read them and researched them in depth and are going to discuss the different methods that have been used. We will also be comparing these methods and discuss the most suitable method for specific cases.

probably the NSF, will have to ask someone in the know

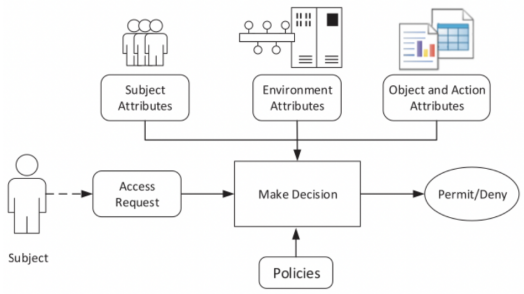


Fig. 1. Basic ABAC model taken from [5]

Figure 1 gives us a general idea of what a basic working ABAC model looks like. The model consists of four main attributes. These include object, environment, subject, and action attributes. Each attribute has its own purpose. The object attribute describes the objects being described like the author, created date, and time-to-live. The subject attribute describes the subject, in this case the user, trying to access the file like their name, role, and department. Environment attribute describes the environment of the access scenario such as the location and time. The fourth and the last attribute, action attribute describes the operation that is being attempted for example, modifying, deleting, appending, etc.

## II. Methods

We surveyed three papers [4]- [6] that presented cryptographic attribute-based access control schemes.

All three papers relied on the use of *bilinear maps* for their operation. In a cryptographic context, bilinear maps are defined in [3] as functions $e : (\mathbb{G}_1 \times \mathbb{G}_2) \to \mathbb{G}_T$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are groups, with the following two properties:

1) Bilinearity: For every $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, and for any integers $a, b$, we have $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
2) Non-degeneracy: If $g_1$ and $g_2$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, then $e(g_1, g_2) \neq 1$, where 1 is the identity element in $\mathbb{G}_T$.

Papers have disagreed on the notation: [4] and [6] require that the two input groups are the same (in the above notation, $\mathbb{G}_1 = \mathbb{G}_2$) and called the input group $\mathbb{G}_0$, while the output group $\mathbb{G}_T$ was called "$\mathbb{G}_1$".

For bilinear maps to be useful, they must be efficiently computable on groups in which the discrete logarithm problem is hard. Practical realizations of this involve using the Weil or Tate pairings on elliptic curves, as shown in [3]. However, for our purposes, we treat these algorithms as proverbial "black boxes" with the desired cryptographic properties.

### A. AHAC-CP-ABE

He et al. developed a scheme called AHAC-CP-ABE in [4], which stands for Attribute-based Hierarchical Access Control, Ciphertext Policy, Attribute-Based Encryption. It uses the Linear Secret Sharing Scheme from [7] to make a matrix $M$ where each row is labeled with an attribute. The sub-matrix consisting of the rows corresponding to a user's attributes gives enough information to reconstruct all and only those keys allowed by the policy.

In pure LSSS, for example, with [4]'s matrix

$$M = \left(\begin{array}{c|ccc} A & 1 & 1 & 0 \\ B & 0 & -1 & 1 \\ C & 0 & 0 & -1 \\ D & 0 & 0 & -1 \end{array}\right) \tag{1}$$

where $A, B, C, D$ are attributes, a user with attributes $B$ and $D$ would be given keys $-k_2+k_3$ (corresponding to the second row) and $-k_3$, which is enough information to deduce $k_2$ and $k_3$. However, a user with only attribute $A$ is given $k_1 + k_2$ and nothing else, which gives no useful information about either key. However, using this scheme would allow users to collude to gain access to resources that neither could access independently. For this reason, AHAC-CP-ABE uses the LSSS as part of a larger system.

AHAC-CP-ABE in [4] begins with a bilinear group system in which $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same group, with a generator $g$. The **Setup** function creates the master key, with random numbers $\alpha$ and $\beta$ used to create the public key $(g, e(g,g)^\alpha, g^\beta)$ and master private key $g^\alpha$. Additionally, for each attribute $u$ a random point $h_u$ in $\mathbb{G}_1$ is chosen and published. (The $h_u$ values are not keys since they are public.)

Users are given their private keys by the administrator. The key depends on the set $S$ of attributes the user has. Importantly, the administrator picks a random value $t$ and sends $(K = g^\alpha g^{\beta t}, L = g^t)$ along with $K_u := h_u^t$ for each attribute $u \in S$. The random $t$ values are used to ensure that no two users are able to collude, since their $K$, $L$, and $K_u$ values are incompatible. Moreover, the $t$ value itself is unknown to the user, and reverse engineering its value requires solving discrete logarithms.

Encryption can be done by anyone with the system public key; it does not need to be the administrator. If $M$ is the LSSS matrix, of dimensions $l \times n$, and $\rho$ is the attribute labels (i.e. $\rho(i)$ is the attribute corresponding to row $i$), then the encryptor creates random numbers $r_1, \ldots, r_l$ and a random vector $\vec{v} = (s_1, \ldots, s_n)^T$ and computes

$$C_j^* = m_j \left(e(g,g)^\alpha\right)^{s_j} \tag{2}$$

$$C_j' = g^{s_j} \tag{3}$$

$$\lambda_i = M_i \vec{v} \tag{4}$$

$$C_i = (g^\beta)^{\lambda_i} h_{\rho(i)}^{-r_i} \tag{5}$$

$$D_i = g^{r_i} \tag{6}$$

for each $j \in \{1, \ldots, n\}$ and $i \in \{1, \ldots, l\}$, and publishes all $2n + 3l$ values along with $M$ and $\rho$.

Decryption is performed entirely by the user, and starts with the user taking the sub-matrix $M_A$ of the user's attribute set $A$, and solving $M_A^T \overrightarrow{\omega_j} = \mathbf{e}_j$ for as many columns $j$ as possible, where $\mathbf{e}_j$ is the $j$-th standard basis vector.

The paper denotes the components of $\overrightarrow{\omega_j}$ by $\omega_{i,j}$, and the set of rows for the user's attributes by $I$. The next step is for the user to compute

$$F_j = \frac{e(C_j', K)}{\prod_{i \in I} \left(e(C_i, L) e(D_i, K_{\rho(i)})\right)^{\omega_{i,j}}} \tag{7}$$

for each $j$ whose $\overrightarrow{\omega_j}$ can be solved for. (Note that while the entire matrix $M$ is public, the values $K_{\rho(i)}$ are unknown to any user who does not satisfy attribute $\rho(i)$, and so $F_j$ will not be computable unless $\omega_{i,j} = 0$.)

The $F_j$ value, if computable, simplifies to $e(g,g)^{\alpha s_j}$ and so deriving "message" $j$ requires dividing the ciphertext by $F_j$.

### B. C-ABAC

The scheme of Zhu et al. in [5] was the most complicated. Their purpose was to make a scheme conforming to a NIST standard for attribute-based access control [8], so the first part of the paper contained terms like "Policy Information Point" and "Policy Decision Point". The actual cryptographic scheme relied on two trusted authorities, the "Policy Center", which creates ciphertexts, and the "Attribute Authories", which generate "Tokens" allowing users to decrypt their files. A graphical overview of the scheme's components is shown in Fig. 2.

In addition to using bilinear maps, this scheme uses the XOR function to mask data: the session key $ek$ was hidden behind an XOR with $e(g^\alpha, h^w)$, where $g^\alpha$ was a public key and the value $g^w$ is given along with the ciphertext; later the encryption algorithm masks the LSSS powers with a second XOR to cancel out nonces. Additionally, its security depended on recomputing the "ciphertext policy" every time a user requests access to a file, because otherwise the nonces $\tau$ would be repeated and users could collude.

This version also requires *five* functions instead of four; after the "ciphertext policy" $C_\Pi$ is generated, the attribute authorities need to generate "Tokens" of the form $Hash(attribute)^{\beta_s/(\beta_s+\tau)}$, where $\tau$ is the nonce and $\beta_s$ is the private key of the object (data file).

The system begins with a public bilinear system where $\mathbb{G}_1 \neq \mathbb{G}_2$, where $g$ and $h$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$

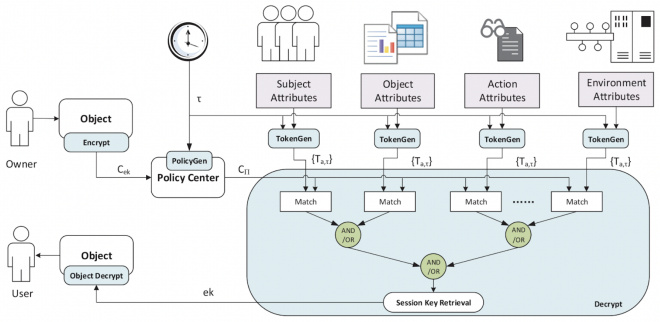


Fig. 2. C-ABAC model taken from [5]

respectively, each with order $p$. Every "entity", including users and data files and the policy center, creates its own public key $g^{\beta_{entity}}$ and keeps $\beta_{entity}$ as the private key. (The policy center's key is denoted $g^{\alpha}$ and $\alpha$.)

Object encryption consists of taking the symmetric key and XOR-ing it with the binary representation of $e(g^{\alpha}, h)^{w}$, where $w$ is randomly chosen from $\{1, \ldots, p\}$. The result is published along with $g^{w}$ to enable decryption.

The next step, called PolicyGen, is to compute the *ciphertext policy* $C_{\Pi}$; only the policy center can do this, since it requires knowledge of $\alpha$. To keep notation as consistent as possible, we will use $i$ for row indices instead of [5]'s $k$. The policy center creates a policy matrix $M$ and a vector $\vec{v} = (t, r_2, \ldots, r_n)$, where here only the first element $t$ is used to decrypt the key. The $\lambda$ values are computed as in [4].

One special feature of this algorithm is the use of *nonces*, which are numbers $\tau$ between 1 and $p-1$ that are changed every time a user issues an encryption or decryption request (which forces PolicyGen to be run again). For each row $i$, the policy center randomly chooses $\gamma_i \in \{1, \ldots, p-1\}$ and looks up the public key $g^{\beta_s}$ of "entity $s$"[1] and computes

$$p_{i,1} = (g^{\beta_s} g^{\tau})^{\gamma_k} \tag{8}$$

$$p_{i,2} = (h^{\alpha})^{\gamma_k} \oplus e(g^{\beta_s}, Hash(\rho(i))^{\gamma_k}) \tag{9}$$

(The $Hash$ function needs to return a point in $\mathbb{G}_2$.) These values, for each $i$ from 1 to $l$, are sent to the user along with $p_0 = (g^{w})^{1/t}$.

However, the presence of $\tau$ requires another component to enable decryption. The user must request "Tokens", one for each attribute $a$ they possess, from the "Token Generation Unit", which sends back the Token $Hash(a)^{\beta_s/(\beta_s+\tau)}$ for each such attribute $a$. The user will then feed $p_{i,1}$ and the appropriate Token for $i$ into the bilinear function $e$ to create a value we will call $\mu_i$ ( [5] used $m_k$, but this is not a message or message key). It turns out that $\mu := \prod_{i \in I} \mu_i^{\omega_i} = h^{\alpha t}$, so since $p_0 = (g^{w})^{1/t}$ is given, the symmetric key can be obtained by

[1]The paper does not make clear what "entity $s$" is. If $s$ is the user decrypting the file, this means the user needs to give their secret key to the Token Generating Unit, since $\beta_s$ will be used directly to compute Tokens in the next step. Also, at least one of $Hash$, $\beta_s$ and $\tau$ must be unknown to the user, or else they could simply create their own tokens for all attributes.

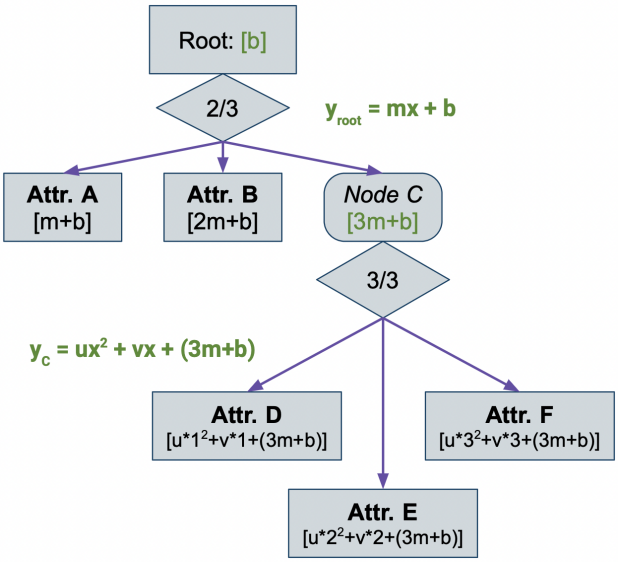


Fig. 3. Example polynomial tree with 5 attributes and 1 internal node.

XOR-ing its masked form (the masking was with $e(g^{\alpha}, h)^{w}$) with $e(p_0, \mu) = e(g^{w/t}, h^{\alpha t}) = e(g^{\alpha}, h)^{w}$. This retrieves the message key successfully.

The advantage of this scheme is that the only coordination between the policy center and attribute authorities required is for the nonces $\tau$; if these are derived from the current time, as [5] suggests, then the two can be otherwise disconnected.

### C. Another CP-ABE

Bethencourt, Sahai, and Waters in [6] used a recursive algorithm for their decryption process. It only takes the security parameter as the input and outputs the public parameters and the master key.

The policy needed here is that of a *policy tree*, where the root node represents the file, the leaf nodes are attributes, and internal nodes (and the root) represent combinations of attributes. Each non-leaf node $x$ has a *threshold* value $t_x$, indicating how many of its children must be satisfied. This is satisfied through the use of secret polynomials $q_x$ at each node $x$, which have a known degree, and satisfied attributes correspond to known values $q_x(i)$ at known inputs $i$ (actually, the user would know $g^{q_x(i)}$). In Fig. 3, nodes A, B, D, E, and F are attributes; node C represents the conjunction of D, E, and F; and the root requires at least 2 of A, B, and C. (As noted in [6], the AND and OR logic gates on $n$ children can be implemented with thresholds of $n$ and 1 respectively; indeed, node C is a 3-way AND gate.)

Note that the $y$-intercept of the polynomial $y_C$ is $3m + b$, which is $q_{root}(3)$ since node C is the 3rd child of the root. If node E were an internal node, its children would have a polynomial with constant term $4u + 2v + 3m + b$.

The setup consists of choosing random numbers $\alpha$ and $\beta$ and publishing a generator $g$ along with $e(g, g)^{\alpha}$ and $h := g^{\beta}$, and also $f := g^{1/\beta}$, which is used in the delegation algorithm. The secret master key is $\beta$ and $g^{\alpha}$.

Encryption in [6] has some similarities to [4], but here, the data owner needs the policy in tree form. The algorithm starts by choosing a secret $s$ and a polynomial $q_R$ (over $\mathbb{Z}_p$) for the root node $R$ such that $q_R(0) = s$ and the degree of $q_R$ is one less than the threshold $t_R$ (since $t_R$ points uniquely define a polynomial of degree $t_R - 1$). Then, the $i$-th child node of $R$ receives the value $q_R(i)$. If any node $x$ has children, then a new polynomial $q_x$ is generated randomly such that $q_x(0)$ is the value at $x$ in a similar process.

The published ciphertext consists of $m \cdot (e(g,g)^\alpha)^s$, $h^s$ (which equals $g^{\beta s}$), and for every leaf node $y$, the values $g^{q_y(0)}$ and $Hash(attr(y))^{q_y(0)}$ (each leaf node corresponds to an attribute).

Users' private keys are created by the administrator. The administrator creates a random number $r$ (modulo $p$, of course) and sends $D := (g^\alpha g^r)^{1/\beta}$. Then, for each attribute $a$ possessed by the user, the admin creates a random $r_a$ and sends $D_a := g^r Hash(a)^{r_a}$ and also $g^{r_a}$.

One property of [6] not found in [4] and [5] is that users can delegate keys, that is, create new keys for any subset of their own attributes. (For example, an employee who has the attributes *Engineering* and *Management* could generate another $\{Engineering, Management\}$ key, or an $\{Engineering\}$ key for a subordinate, without invoking the administrator.) If $\tilde{r}$ and $\tilde{r}_a$ are new random values (with one $\tilde{r}_a$ for each attribute $a$ in the subset), the user sends $\tilde{D} = D \cdot f^{\tilde{r}}$, and for each attribute $a$ in the subset, $\tilde{D_a} = D_a g^{\tilde{r}} Hash(a)^{\tilde{r}_a}$ and $g^{\tilde{r}_a}$. Note that the effective $r_{new}$ of the new key is actually $r + \tilde{r}$, and the first summand is unknown.

Since the tree construction and encryption is recursive, the decryption algorithm presented initially is recursive as well (although the authors did construct an improved method to decrypt the entire tree at once). The decryption process generates the value $F_x := e(g,g)^{r q_x(0)}$ at each node $x$ recursively:

- If $x$ is a leaf node with attribute $a$, then
$$F_x = \frac{e(D_i, C_y)}{e(g^{r_a}, Hash(a)^{q_x(0)})} \tag{10}$$
assuming that $g^{r_a}$ is known (as part of the user's private key); otherwise $F_x = \text{NULL}$.
- If $x$ is a non-leaf node, then its threshold value $t_x$ is publicly known. First decrypt $t_x$ child nodes of $x$ recursively (call this set of nodes $S_x$ and the set of indices $S'_x$), then compute
$$F_x = \prod_{z \in S_x} F_z^{\Delta_{index(z), S'_x}} \tag{11}$$
where $\Delta_{i,S}$ is the Lagrange polynomial[2] defined by
$$\Delta_{i,S}(x) = \prod_{s \in S, s \neq i} \frac{x - s}{i - s} \tag{12}$$
so that $\Delta_{i,S}(i) = 1$, and if $s \in S$, $\Delta_{i,S}(s) = 0$. (See [6], p. 6, for a proof that the $F_x$ computation works.) If there are not $t_x$ children that are successfully decrypted, then $x$ cannot be decrypted either, and $F_x = \text{NULL}$ again.

Recursively decrypting the root node $R$ in this way produces $e(g,g)^{r q_R(0)} = e(g,g)^{rs}$. The final step consists of computing
$$\frac{(m \cdot e(g,g)^{\alpha s}) e(g,g)^{rs}}{e(h^s, D)} = \frac{m\, e(g,g)^{\alpha s + rs}}{e(g^{\beta s}, g^{(\alpha + r)/\beta})} \tag{13}$$
which simplifies correctly to just $m$, the "message".

[2]For instance, $\Delta_{3,\{3,4,5\}}(x) = \frac{x-4}{3-4}\frac{x-5}{3-5}$.

## III. Comparison

Figure 4 summarizes the main properties of each system. LSSS refers to the use of the Linear Secret Sharing Scheme. Message hiding refers to the mechanism by which the message (or symmetric key) itself is encrypted: [4] and [6] convert it into a point on $\mathbb{G}_T$ and multiply it by another value, while [5] converts it into binary and uses the XOR function on another value.

| **Property** | [4] | [5] | [6] |
|---|---|---|---|
| LSSS | Yes | Yes | No |
| Polynomials | No | No | Yes |
| Dynamic | No | Yes | No |
| Message hiding | $\mathbb{G}_T$ op.[a] | XOR | $\mathbb{G}_T$ op.[a] |
| Multiple file groups at once | Yes | No | Maybe |
| Security assumptions | q-BDHE | CDH<br>SDH<br>eBDH<br>eGDHE | None |
| Key delegation | No | No | Yes |

[a] "$\mathbb{G}_T$ op." means converting the message to a point in $\mathbb{G}_T$ and multiplying it by another point obtained from the bilinear function.

Fig. 4. Properties of each encryption system.

### A. General Comparison

Each paper has its own unique strength that no other paper supports, although it may be possible to adapt some of them to acheive these goals.

Papers [4] and [6] are very similar – both encrypt messages (or keys) by computing $M\, e(g,g)^{\alpha s}$. The polynomials of [6] can be solved with matrix algebra, but there is also another algorithm that works specifically for polynomials. However, [4] claims to be more efficient than [6] because the latter uses a tree structure directly in the algorithms, while in [4] the hierarchical tree is converted into a matrix.

The scheme in [5] is wasteful since you have to generate the ciphertext policy (PolicyGen) differently for each user who requests files. However, this is also the only scheme that allows attributes to be changed and revoked without changing the private keys.

But [5] has the advantage of being dynamic, that is, it can respond to changes in policy and user attributes over time without changing users' keys. This means that the system could have an attribute "hired after March 2020" in the policy, and the attribute authority decides whether or not to give the

token. With [6] (or [4]), the system has to store the hire date as a binary integer of attributes, and build "after" out of ANDs and ORs. (Or, if there are only a few cutoff dates, it could use attributes for each date.)

Scheme [4]'s main advantage is that it allows multiple files (or collections of files) to be encrypted with the same access tree. This does not work in [5], since the ciphertext policy must be created anew for each decryption request. Paper [6] does not claim to allow this either. However, [6]'s decryption process results in computing the quantity $e(g,g)^{rq_x(0)}$ for each satisfied node $x$, which ultimately results in $e(g,g)^{rs}$ at the root. The ciphertext object depends on $e(g,g)^{\alpha s}$ and $h^s$, so it would be possible to simply have other ciphertexts that depend on $\overline{s} = q_x(0)$ for other internal nodes $x$. Unfortunately, as [4] points out, [6] "cannot provide strong security" ( [4], p. 3) since [6] only provides a "security intuition" ( [6], p. 6), while [4] is designed around a concrete security assumption, namely, the $d$-parallel bilinear Diffie-Hellman assumption ( [4], p. 12).

The biggest advantage of [6] is the key delegation, where users can make a sub-key of their key to give to their subordinates. Although such users could act maliciously (by, for instance, picking identical $\tilde{r}$ values) to increase the access of the subordinates, this can only work within the attacker's set $S$ of attributes. This attack is no stronger than the attacker simply sharing their own key directly.

Finally, we note that polynomials, as used in [6], are a special case of matrices since you can represent the polynomial by using the matrix

$$\begin{pmatrix} 1 & 1 & \dots & 1^n \\ 1 & 2 & \dots & 2^n \\ 1 & 3 & \dots & 3^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & r & \dots & r^n \end{pmatrix} \tag{14}$$

However, this does not make [6] identical to a single-file version of [4] since the public keys and ciphertexts have different forms.

### *B. Performance Comparison*

In [4], the researchers compared the encryption and decryption time as the number of attributes increase with three other systems. They found that both the encryption and decryption time is the fastest compared to the three other systems. Both encryption and decryption increase linearly but decryption of the model was faster compared to the encryption time. In this paper ( [4]), one of the models that they compared to was our third research paper ( [6]). This model still was faster (encryption and decryption time) than the CP-ABE model discussed in [6].

In [5] the researchers compared the five algorithms for the functions as the number of attributes increases. These five are: Setup, Encrypt, Decrypt, PolicyGen, and TokenGen. Setup and Encrypt were roughly constant and did not change drastically. TokenGen, decrypt, and PolicyGen all increase linearly. TokenGen increases more than the other two and decryption is more efficient compared to the other two.

In [5], the key generation time which was a part of the setup algorithm, stayed constant, whereas in [4], the private key generation time increased slowly as the number of attributes increased.

In [6], they also evaluated the private key generation time as the number of attributes increased. In this paper, the time increased as the number of attributes increased.

## IV. Conclusion

We have surveyed three implementations of cryptographic attribute-based access control. We have discovered that bilinear cryptography is frequently used in this domain to allow for data decryption without collusion between users. After evaluating the three different papers and their private key generation times, we observed that the only model that had an almost constant graph for time as the number of attributes increased was [5]. The other two increased linearly. However, [5]'s need for repeatedly calling the PolicyGen function makes it extremely inefficient; this is not shared by [4].


## Acknowledgments

This research was supported by the National Science Foundation REU program. The authors would also like to thank Dr. Feng Li, Dr. Xukai Zou, and Sheila Walter for their support throughout the program.


## References


[1] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo and J. Voas, "Attribute-Based Access Control," in Computer, vol. 48, no. 2, pp. 85-88, Feb. 2015, doi: 10.1109/MC.2015.33.

[2] R. S. Sandhu and P. Samarati, "Access control: principle and practice," in IEEE Communications Magazine, vol. 32, no. 9, pp. 40-48, Sept. 1994, doi: 10.1109/35.312842.

[3] B. Lynn, "On the implementation of pairing-based cryptosystems," PhD diss., Stanford University, 2007.

[4] H. He, Lh. Zheng, P. Li et al, "An efficient attribute-based hierarchical data access control scheme in cloud computing," in Hum. Cent. Comput. Inf. Sci. 10, 49 (2020), doi: 10.1186/s13673-020-00255-5.

[5] Y. Zhu, R. Yu, D. Ma and W. Cheng-Chung Chu, "Cryptographic Attribute-Based Access Control (ABAC) for Secure Decision Making of Dynamic Policy With Multiauthority Attribute Tokens," in IEEE Transactions on Reliability, vol. 68, no. 4, pp. 1330-1346, Dec. 2019, doi: 10.1109/TR.2019.2948713.

[6] J. Bethencourt, A. Sahai and B. Waters, "Ciphertext-Policy Attribute-Based Encryption," 2007 IEEE Symposium on Security and Privacy (SP '07), 2007, pp. 321-334, doi: 10.1109/SP.2007.11.

[7] A. Beimel, "Secure schemes for secret sharing and key distribution," PhD diss., 1996.

[8] NIST ABAC standard. Better citation format needed. https://csrc.nist.gov/projects/attribute-based-access-control