

Authenticating with Attributes

Abstract. This paper proposes an attribute based authentication scheme (AAS). An AAS scheme is a digital signature scheme that allows a verifier to decide on the set of attributes (s)he would like the signer to possess. The verifier sends the request to a group of possible signers as a monotone boolean expression. Any member with sufficient attributes can sign. This paper defines such a scheme and its security notions.

1 Introduction

The idea of attribute based authentication is proposed to serve the following scenario:

Scenario 1 *Bob manages a company with many departments and many employees. Employees are divided according to positions, responsibilities, levels, etc. Alice as a verifier, wants the employee signing to prove to be a senior manager in department A or a manager (senior/junior) in department B. There is a private key for each employee and a public key for the company used for the purpose of signing and verifying. Additional to that each employee obtains from the company a registration key that will help him get attribute related keys from attribute authorities. Attribute authorities can be the different departments of the company. Alice sends her request to the whole company. The request format should make it clear that Alice needs the signer to be: (Senior manager and Department A) or ((Junior Manager and Department B) or (Senior Manager and Department B)). Employees with sufficient attribute-related keys can sign on behalf of the company.*

The scheme should include the following properties:

- Unforgeable: It is hard for someone outside the group of possible signers to forge a signature. The signer has to be an employee in the company. In addition to that, proof of possession of attributes is hard to fake. Only employees with sufficient attributes can create valid signatures.
- Anonymity of Identity: Signatures can not be linked to a signer. The verifier or any eavesdropper can not reveal the identity of the signer from the signature. However, anonymity can only be subject to the required attributes. For example, if there is only one person (say Joe) with a given attribute X , and Alice requests the attribute X , the signer cannot be anonymous.
- Unlinkable: The verifier cannot distinguish whether two signatures were created by the same signer or not. If signatures are linked, the identity of the signers may get exposed by time causing anonymity to break.
- Traceable: Traceability is required in order to prevent employees from misusing anonymity. Bob and only him as a manager can revoke the anonymity of the signers and trace signatures to members of the group even if they were part of a coalition.
- Anonymity of Attributes: The scheme should not enclose the attributes with the signature, it should just provide a proof of possession of sufficient set of attributes. For instance, in Alice's policy she does not need to know whether the employee signing is actually in department A or B. All she needs to know is that her policy is satisfied and it does not matter how.
- Coalition Resistant: If two valid employees have enough attributes, jointly, that satisfy the policy they should not be able to create a valid group signature as if they were one person. For instance, an employee in department A but not a senior manager and a senior manager in department C should not be able to create an acceptable signature to Alice.

Different attribute oriented authentication schemes exist in literature, however each of them is designed to serve a certain application. The properties we listed above never co-existed in one scheme. The following section defines the scheme and its algorithms.

2 AAS Definition

An attribute authentication scheme consists of four entities; a central authority (Bob), an attribute authority (department), a signer (employee) and a verifier (Alice). The central authority will publish

one public key base for the group (Step 1a Figure 1) and this will be used in verifying a signature. The central authority creates many pairs of private key bases and registration keys to be used in signing messages. Each member (possible signer) of the group will be given their unique pair (i.e. Bob gives each employee a pair as shown in Step 1b Figure 1). Each member will be using the registration key to register with an attribute authority (Step 2 in Figure 1). Registering with an authority implies getting a copy of a private attribute key (Step 3a in Figure 1). A set of private attribute keys will then be used in signing a message. The public key base created earlier will be used by the attribute authority to create an attribute public key (Step 3b in Figure 1). The verifier (Alice) will create a verification key using the public key base and as many public attribute keys as needed (Step 4 in Figure 1). The verification key describes what attributes the verifier would like the signer to possess. It will also be used in both signing and verifying processes. An employee with such qualification (i.e has enough private keys) can then sign a message (Step 5 in Figure 1). The system can have many attribute authorities but has to have one central authority. The central authority can be an attribute authority itself.

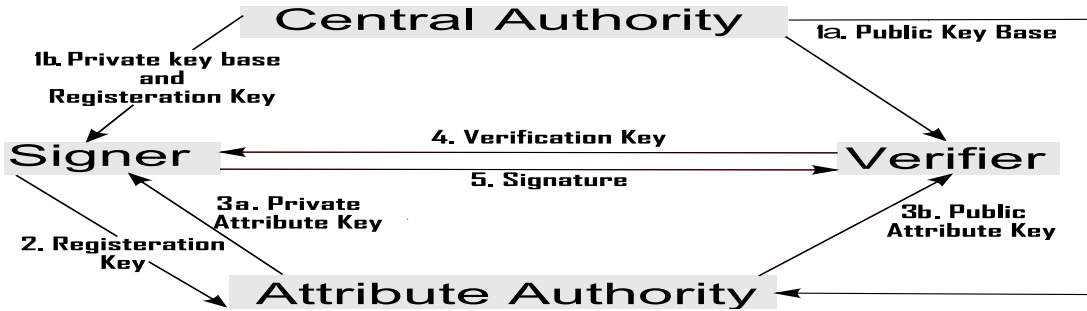


Fig. 1. Attribute Authentication Scheme

2.1 AAS Scheme

An attribute authentication scheme (AAS) contains a suite of algorithms and protocols that are executed by the different entities. To describe the scheme we define the following algorithms and protocols:

- **Setup(k)**: This algorithm is run by the central authority, it takes a security parameter k as an input and outputs two sets of parameters, S_{pri} and S_{pub} . The system parameters S_{pri} is kept by the authority, while S_{pub} is published for all to see and use.
- **M.KeyGen(S_{pri}, S_{pub})**: This algorithm is run by the central authority. It takes the inputs S_{pri} , and S_{pub} . It then generates pairs of private key bases $bsk[i]$ and registration keys A_i where the pairs are distributed to the members of the group. Then S_{pub} and S_{pri} are used to generate a public key base w known to all. The $bsk[i]$ is kept private to the user, while the A_i is given to trusted third parties, as shown in $A.KeyGen_{pri}$. Later on, in this chapter, we will show how to replace this algorithm with a protocol that conceals A_i from everyone but the member.
- **A.KeyGen_{pub}(S_{pub}, w)**: This algorithm is run by the attribute authorities. Each authority is responsible for one or more attributes and for every attribute j , the authority creates a corresponding master key t_j . It is this key which is used to create the attribute-related keys. Using the master keys, the public parameters S_{pub} and public key base w , the authority creates public keys bpk_j representing the attributes it supports. Only the attribute authorities can produce such keys since it requires the knowledge of t_j .
- **A.KeyGen_{pri}(A_i, j, RL)**: This algorithm is run by the attribute authorities. In this algorithm member i registers with his key A_i to obtain a special private key $T_{i,j}$. Due to the fact that the attribute authority is supposed to be independent from the central authority it is best to check the user against a list RL , where RL contains all registration keys A_i of all users which have been revoked. If user has not been revoked, the private attribute key is calculated using the attribute authority's master key t_j . Member i will be then using his private key $gsk = (bsk[i], A_i, T_{i,1}, \dots, T_{i,\mu})$ to sign. We should point out that not all $T_{i,j}$ have to be generated by the same attribute authority and that each signature will have a different set of $T_{i,j}$ depending on the verifier's request. We will

exchange this algorithm with a protocol that hides A_i from the attribute authority and gets the same output of this algorithm .

- **Verifign**($\mathcal{U}_i(\mathbf{gsk}, \mathbf{M}), \mathcal{V}(\mathbf{M}, \overline{\mathbf{B}}, \mathbf{w})$) : This stage is a protocol between the verifier \mathcal{V} and the signer \mathcal{U}_i . The verifier takes as an input $\overline{B} = (bpk_1, \dots, bpk_m)$ and a message M to create a verification key \overline{D} to be sent to the group. The signer uses his private key gsk , the message M and \overline{D} as inputs to an algorithm **Sign** that creates a signature σ that corresponds to \overline{D} as follows $\sigma = \text{Sign}(gsk, M, \overline{D})$. \mathcal{U}_i sends σ to \mathcal{V} and the verifier runs an algorithm **Verify** using σ , M , \overline{D} and w as inputs. The algorithm checks the validity of the signature and whether or not the signer is revoked as follows $\text{Verify}(\sigma, M, \overline{D}, w, RL) = \{\text{Accept}, \text{Reject}\}$.
- **Revoke**($\mathbf{A}_i, \mathbf{RL}$) : This algorithm is run by the central authority. It adds a registration key A_i of revoked users to the revocation list RL .
- **ChkRvk**(σ, \mathbf{RL}) : This algorithm takes a signature σ and a revocation list RL . It returns i if the user is revoked and on the list otherwise it returns -1 . $ChkRvk$ is an algorithm run by either the verifier or the central authority. The index returned does not mean much to the verifier. The verifier only wants to know whether the user has been revoked or not. So if the value of the index is anything other than -1 the user has been revoked. If $ChkRvk$ is run by the central authority then the index refers to a user and that will help in tracing signatures to users as explained in the following *Open* algorithm.
- **Open**(σ, \mathbf{FRL}) : This algorithm is meant for tracing a signer in case of a dispute and revoking his anonymity. FRL is a list created by the central authority who stores the value A_i for all members of the groups and adds all members (while maintaining indexes) to that fake revocation list FRL . It runs $ChkRvk(\sigma, FRL)$ which returns an index i . Since all members belong to FRL the value of i must reveal the identity of the signer. *Open* is not a totally new algorithm because it uses $ChkRvk$ on a fake list FRL .

2.2 Security Notions

Three game models are enough to cover the properties we mentioned earlier: “full anonymity” model, the “full traceability” model, and the “unforgeability of attributes” model. The three models allow the adversary to query certain oracles run by the challenger. In the full anonymity and full traceability game model we assume that the attribute authority is corrupted and we give the adversary the ability to choose the master keys he would like to be challenged on.

Before we define the adversarial models we list the oracles:

- **USK Oracle**: To query this oracle the adversary sends the challenger an index i in order to find the user’s private key base $bsk[i]$ and registration key A_i . The challenger will send the pair $(bsk[i], A_i)$ to the adversary. Note that since the adversary is given the ability to choose the attribute master keys he would like to be challenged on, he can create private attribute keys from whatever private key bases he has.
- **Signature Oracle**: To query this oracle the adversary sends the challenger a verification key he created \overline{D} , a message M and an index i . The adversary requires the signature of member i on message M using \overline{D} . The challenger sends σ
- **Revoke Oracle**: To query this oracle the adversary sends the challenger an index i . The challenger adds A_i to the list RL . The list RL is public and the adversary can access it.
- **AttPriKey Oracle**: To query this oracle the adversary sends a registration key A_i to the challenger together with an index of the attribute j . The output of this query is a private attribute key $T_{i,j}$.
- **AttMasKey Oracle**: To query this oracle the adversary sends an index j to the challenger. The challenger responds with sending the output t_j .

The adversary initializes both the traceability and anonymity game models by deciding the universal set of attributes U in which he would like to be challenged upon. Assume U is of size m and contains a list of master keys t_j that will be used. Both challenger and adversary have access to U .

AAS Full Anonymity: We say that an AAS Scheme is fully anonymous under a specific set of attributes, if no polynomially bounded adversary has a non-negligible advantage against the challenger in the following AAAS game:

- **AAAS.Setup**: The challenger plays the role of the central authority. He runs the algorithms *Setup*, and *M.KeyGen* to produce the systems S_{pub} , and S_{pri} . He also generates the n private key bases $bsk[i]$ and n registration keys A_i . He sends to the adversary parameters S_{pub} . Finally, both can generate m public attribute keys $\langle bpk_1, \dots, bpk_m \rangle$.

- **AAAS.Phase (1):** The challenger runs the three oracles, USK, Signature oracle, and Revoke oracle as explained above. The adversary can query these oracles to obtain any information he thinks he will require in breaking the scheme.
- **AAAS.Challenge:** The adversary asks to be challenged on a message M , two indexes i_0, i_1 , and verification key \bar{D} . The challenger responds back with a signature σ_b , where $b \in \{0, 1\}$. The signer can be either i_0 or i_1 . Both i_0, i_1 should not have been queried in the Revoke oracle or USK oracle, however it can be queried on the Signature oracle.
- **AAAS.Phase (2):** This stage is similar to Phase 1. Except that (i_0, i_1) should not be sent to the Revoke nor the USK oracles.
- **AAAS.Output:** The adversary outputs a guess $\bar{b} \in \{0, 1\}$. If $\bar{b} = b$, *Adam* wins the game.

The adversary has been given strong attack capabilities by getting access to oracles such as USK, Revoke and Signature. He does not need to query oracles *AttMasKey* and *AttPriKey* because he obtains all master keys t_j . He is not allowed to query both i_0 , and i_1 in the Revoke oracle or the USK oracle because that will give away the identities. If one of them is revoked (*ChkRvk* is used) or if he has the private key base then the adversary can distinguish the signature and identify the signer. However, the adversary can query signatures of (i_0, i_1) and still his advantage in guessing the signer should be negligible. This implies that the signatures can not be linked and the advantage of winning the game is $Adv_{AAAS}(n, k) = Pr[b = \bar{b}] - 1/2$, where n is the maximum bound of the numbers of members and k is the security parameter used for setting up the system.

Definition 1. Full Anonymity:

A scheme is fully anonymous if for all polynomial time adversary, $Adv_{AAAS}(n, k) < \varepsilon$ and ε is negligible.

AAS Full Traceability: We say that our AAS scheme is traceable if no polynomially bounded adversary has a non-negligible advantage against the challenger in the following *TAAS* game:

- **TAAS.Setup:** The challenger plays the role of the central authority as done in the setup stage of the anonymity game model mentioned earlier. He runs the algorithms *Setup*, and *M.KeyGen* to produce S_{pub} , and S_{pri} . The challenger also generates n private key bases $bsk[i]$ and n registration keys A_i . S_{pub} are sent to the adversary and he is also given all registration keys A_i . Finally, both parties can generate m public attribute keys $\langle bpk_1, \dots, bpk_m \rangle$.
- **TAAS.Queries:** The challenger runs two oracles, USK oracle and Signature oracle, that the adversary can query.
- **TAAS.Output:** The adversary asks to be challenged on a message M which he sends to the challenger. The challenger calculates a new \bar{D} and sends it back to the adversary. The adversary replies with a signature σ . The challenger verifies the signature. If it is not valid return 0. If it turns out to be a valid signature the challenger tries tracing it to a signer. If it traces to a signer in which the adversary did not query before or if it traces to a nonmember then the adversary wins the game. The challenger returns 1 if the adversary wins else 0 is returned.

Note that the adversary does not have to query the Revoke oracle. He has a more powerful capability and that is the list of all registration keys. The adversary can run the revoke algorithm himself. The adversary does not need to query oracles *AttMasKey* and *AttPriKey* because he obtains all master keys t_j . Full traceability includes unforgeability. One can reduce the challenge to produce a valid pair of message and signature, where the message was not queried in phase 1. The adversarial model with such reduction is the definition of unforgeability. Therefore full traceability implicitly proves unforgeability. Additionally a strong formalization of coalition resistance is obtained in this game since this can be defined with a similar game where the adversary is not given the registration keys. If we refer to the returned value as *Exp* then the advantage of winning the game is notated as $Adv_{TAAS}(n, k) = Pr[Exp = 1]$.

Definition 2. Full Traceability:

A scheme is fully traceable if for all polynomial time adversary, $Adv_{TAAS}(n, k) < \varepsilon$ and ε is negligible.

AAS Unforgeability of Attributes: We say that our AAS scheme is Attribute-Unforgeable if no polynomially bounded adversary has a non-negligible advantage against the challenger in the following *AFAAS* game:

- **AFAAS.Setup:** The Challenger plays the role of the central authority and all attribute authorities in the system. He runs the algorithms *Setup*, and *M.KeyGen* to produce S_{pub} , and S_{pri} . The challenger also generates n private key bases $bsk[i]$ and n registration keys A_i . The challenger can

produce the set of all attributes in the system since he is playing the role of all attribute authorities. He generates a set of master keys t_1, \dots, t_m . He creates all public attributes needed bpk_1, \dots, bpk_m . The adversary is given S_{pub} , the list of registration keys A_i , and the set of attribute public keys bpk_j . The challenger keeps the list of private keys $bsk[i]$, parameters S_{pri} and the list of master keys t_j .

- **AFAAS.Phase (1):** The challenger runs the oracles Signature, USK, AttPriKey and AttMasKey while the adversary can query any of them. The adversary does not need the Revoke oracle since he has all registration keys A_i and can trace signatures using the open algorithm.
- **AFAAS.Challenge:** The adversary sends a tree T_1 , user l and attribute z in which he would like to be challenged on. The challenger replies with \bar{D} for a tree T_2 where T_2 has two subtrees the first is T_1 and the other is based on t_z . The threshold value of the root in T_2 is 2. The challenge condition is that user l has not been queried in AttPriKey for the attribute z . Furthermore the challenged index z should not have been queried in AttMasKey. These two conditions are reasonable as violating them would contradict the purpose of the game.
- **AFAAS.Phase (2):** This phase is similar to Phase 1 as long as the challenge conditions are not broken.
- **AFAAS.Output:** The adversary outputs a signature σ for the user l on the verification key \bar{D} . If that signature is valid then the adversary wins and the challenger outputs 1 otherwise the adversary loses and the challenger outputs 0.

The adversary can trace any signature since he has all registration keys. He can corrupt users since he can query the USK oracle. He can corrupt attribute authorities by obtaining the master keys with the AttMasKey oracle. The challenge is to create a signature using an attribute z that has not been queried in AttMasKey and a signer l that does not have the attribute private key for z . If the adversary can output a valid signature that proves that user l has attribute z he would win the game. If we refer to the returned value as Exp then the advantage of winning the game is notated as $Adv_{AFAAS}(n, k) = Pr[Exp = 1]$

Definition 3. Unforgeability of Attributes:

An AAS scheme is attribute-unforgeable if for all polynomial time adversary Adam, $Adv_{AFAAS}(n, k) < \varepsilon$ and ε is negligible.

3 Conclusion

Recently attribute based cryptography became an important research line [1, 3, 4, 2]. In this paper we introduce attribute based authentication scheme. We explain its algorithms and protocols. We later define the security notions of “Full Traceability”, “Full Anonymity” and “Unforgeability of Attributes”.

References

1. V. Goyal, O. Pandey, A. Sahaiz, and B. Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.
2. H. Maji, M. Prabhakaran, and M. Rosulek. Attribute-Based Signatures: Achieving Attribute-Privacy and Collusion-Resistance. Cryptology ePrint Archive, Report 2008/328, 2008. <http://eprint.iacr.org/>.
3. R. Ostrovsky, A. Sahai, and B. Waters. Attribute-based Encryption with Non-Monotonic Access Structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.
4. B. Waters. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. Cryptology ePrint Archive, Report 2008/290, 2008. <http://eprint.iacr.org/>.