NTNU – Trondheim
Norwegian University of
Science and Technology

Functional Key Exchange In A Distributed Environment

Anders Kofoed

Submission date: October 2014
Responsible professor: Colin Boyd, ITEM
Supervisor: Colin Boyd, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Functional Key Exchange In A Distributed Environment
Student: Anders Kofoed

Problem description:

Functional encryption is a new generalization of public key cryptography which allows very flexible access control to encrypted data. It is natural to consider extensions of the functional idea to other cryptographic primitives. One such primitive is key exchange. Since the scheme provides a much more dynamic and flexible way of exchanging secrets, it is natural to consider systems where users join and leave dynamically - such as chat rooms, Internet forums or similar distributed systems.

This project will explain functional key exchange and compare the technique to traditional group key exchange mechanisms, considering both efficiency and security properties. It will contain a prototype implementation of an attribute based authenticated key exchange scheme in a distributed environment, based on existing work done on attribute based key exchange - using the Charm framework and Python. Different application areas for such a system will be explored and problems arising discussed.

Responsible professor: Colin Boyd, ITEM
Supervisor: Colin Boyd, ITEM

Abstract

Write abstract

Preface

write abstract

Contents

Acronyms

AB-AKE attributed-based authenticated key exchange.

ABE attribute-based encryption.

AKE authenticated key exchange.

BDDH bilinear decisional Diffie-Hellman problem.

CA certificate authority.

CDH computational Diffie-Hellman problem.

DDH decisional Diffie-Hellman problem.

DEM data encapsulation mechanism.

IB-AKE identity-based authenticated key exchange.

IBE identity-based encryption.

KEM key encapsulation mechanism.

KMS key management service.

LSSS linear secret-sharing scheme.

NTNU Norwegian University of Science and Technology.

PBKE predicate-based key exchange.

PKI public key infrastructure.

Chapter 1
Introduction

1.1 Motivation

In distributed systems users might not want to publish their identity if it isn't absolutely necessary, which in most cases it isn't. It is more important that the user is legit and have the correct privileges. If we could assure that all users participating in some protocol or application had the right privileges or simply the correct purpose, we could go without knowing the exact identities. This can be achieved using attributes in the encryption mechanisms. Attributes can be anything - personal attributes as birth year, gender or nationality. An example could be affiliation to user groups providing access control, where only the group members would be allowed to communicate. Or attributes could be related to certifications or titles, this way you can be assured that the person you are communicating with has knowledge about something you need help with, without having to knowing the identity. This generalisation can also be extended to include systems where the identity is one of or the only attribute. Thus achieve identity based systems without the need to generate, distribute and store huge amounts of public keys. This project describe the term functional key exchange, covering key exchange mechanisms using the mentioned ideas to achieve dynamic group key exchange in a functional manner. Not exposing identities if not absolutely necessary is the main focus when discussing possible applications of these functional key exchange methods. There already exist several proposed schemes for both attribute- and identity-based key exchange, but there are few or none examples of systems applying the schemes in real applications.

1.2 Related work

1.3 Scope and objectives

The project will be focused around applying a attribute-based key exchange scheme in a real application, based on a implementation of simple attribute-based encryption in the Charm framework [2].

1.4 Limitation

The project will not try to solve problems related to key distribution and how to deploy a secure and trustworthy key management service. For the proposed system it will be assumed that this kind of service exists.

1.5 Outline

The background chapter will describe all the components that are essential to both functional encryption and key exchange, from basic public key encryption to functional encryption schemes, including secret sharing, pairing-based encryption and key encapsulation. The functional encryption algorithms will be presented and discussed using code from Charm, this is logical since one of these schemes is basis for the system implemented in this project. In the next chapter some functional key exchange schemes will be described and possible application areas discussed. Finally a prototype system will be implemented and presented, using attribute-based encryption and key encapsulation. The system will be a distributed chat using attribute-based key exchange to achieve secure communication with dynamic users.

Chapter 2
Background

This chapter will go through the material from which the applications and implementations later rely. Going through some of the most important schemes and protocols both in encryption and for key exchange.

2.1 Public Key Encryption

Public key cryptography or asymmetric cryptography allows encryption of messages without the parties sharing a secret key. Each user have a pair of keys, one public and one private key. The private key is used to decrypt and sign messages for authentication, while the public key - known publicly - is used to encrypt messages.

– Setup - takes the security parameters, depending on the implementation, and outputs a public/private key pair.

– Encryption - takes the public key of the receiver as input together with the message, outputs a cipher text. Only the corresponding private key can decrypt.

– Decryption - uses the private key to decrypt the message.

This setup can also be used to achieve message authentication using digital signatures. Either by the use of the same alogrithms as mentioned above or with separate ones besides these. A common problem with public key crypto systems is how you are supposed to trust that a given public key used to sign something really belongs to the claimed owner. This issue is normally dealt with using a trusted certificate authority (CA) issuing certificates binding a public key to a user identity. The CA has a public/private key pair, which is used to sign certificates on request. Initially the CA have to publish its public key together with a CA certificate. This is stored by in the browsers of users trusting it. A user will typically send a signed certificate request to the CA, including the identity to be associated with the public key. The CA verifies the request signature and produces a certificate which is signed using the CAs own public key. When signing, the user will include this certificate as profe of his identity, which can be verified by anybody in possession of the CAs public key and CA certificate.

Pgp/Gpg [14, 15] is an extension to the classic public key scheme, combining the speed of symmetric key cryptography with the dynamic nature of public key systems. This is done by generating a random session key which is used to encrypt a message, this key is then encrypted using the public keys of each recipients and concatenated together with the cipher text. Pgp uses a hierarchy of trusted CAs as described, but also what's' called a "web of trust" where users can sign the public keys of eachother to assure authenticity.

2.2 Secret Sharing

In many cases it may be desirable to use more than one key when encrypting, and later require a subset of these when decrypting. This concept is called secret sharing or secret splitting - a secret is "divided" into n pieces which then is distributed. To recover the message k of these n pieces needs to be present. Shamir proposed a solution to this with his secret sharing scheme [18]. The main idea behind the scheme is that given k points in a plane $(x_1, y_1), ...(x_k, y_k)$ there is only one polynomial $y(x)$ of degree $k - i$ such that $y(x_i) = y_i$ for all i. We can thus chose a prime p, larger than the desired number of shadows, and a random number $a < p$. Then equation 2.2 is the polynomial used, with the secret m. The shadows are $(x, y(x), p), x = \{1, 2, ..., n\}$ of which only k is needed to recover m. With k shadows we obtain a set of k equations with k unknown which is solved unambiguously.

$$y(x) = m + a_1 * x + a_2 * x^2 + ... + a_{k-1} * x^{k-1} \tag{2.1}$$

2.2.1 Linear Secret Sharing

A linear secret-sharing scheme (LSSS) ...

write about linear secret sharing - used in the implementation of ABE in charm

2.3 Pairing-based encryption

Let G be a group of prime order q with generator g. $x, z, y \in \mathbb{Z}_q$. The *Discrete-logarithm* problem says that it is hard it is hard to obtain x from g^x and g. Further we have the *computational Diffie-Hellman problem (CDH)* which extends this saying that it is hard to calculate g^{xy} from the tuple (g, g^a, b^b). Finally we can say that it is hard to determine if $z = xy$ given (g, g^x, g^y, g^z) - *decisional Diffie-Hellman problem*

(DDH). These assumptions are used, and relied on, in earlier crypto systems. The systems explored in this project use pairing of the traditional group assumptions. The idea behind this is to use a mapping between two cryptographic group which allows the creation of new schemes based on the reduction of one of the problems from earlier. The most renowned pairing-based construction is the *bilinear map*. G_1 and G_2 are groups of prime order q. If $e : G_1 \times G_1 \rightarrow G_2$ then the mapping e should have the following properties to be useful:

- Bilinearity - for all $P, Q \in G_1$ and all $a, b \in \mathbb{Z}_q$, then $e(P^a, Q^b) = e(P, Q)^{ab}$
- Non-Degeneracy - if $P \neq 0 \implies (P, P) \neq 1$

The Weil and Tate pairing are the most used pairings where these properties hold. The pairings usually consist of one Elliptic curve paired with a finite field. The point with all this is that problems in the first group may not all be hard in the pairing group. Discrete-logarithms are still hard since $e(g, g)$, $e(g, g^a) \in G_T$ is as hard as $g, g^a \in G$ where G_T is the pairing group. For DDH though, we can see that to test if $z = xy$ given g, g^x, g^y, x^z, we can just check if $e(g, g^z) = e(g^x, g^y)$. DDH is thus replaced by bilinear decisional Diffie-Hellman problem (BDDH) which is defined as - given $h, g, g^x, g^y, e(h, g)^z$ it is hard to decide if $xy = z$. [11] These definitions made it possible to implement the identity-based encryption (IBE) and attribute-based encryption (ABE) as described in the next section.

not sure if this is a precise understanding of pairings(?)

2.4 Functional Encryption

Before public key cryptography was introduced, secure communication was only achieved when two parties possessed the same secret key which could be used to encrypt and decrypt messages between them. This was changed with the introduction of public key cryptography as mentioned in 2.1. Now, many years later, this is no longer sufficient to cover the new notion of the Internet - with distributed systems and cloud-based services. In standard public key encryption schemes the focus is on full encryption and decryption of the message, it's all or nothing, either you get to know the plain text or you can't. Functional encryption, as described in [6], describes a different perspective on cryptography where what you get to know can differ depending on who or what your are. One key might be used to decrypt only a specified part of the cipher text - a "function" of it. Functional schemes also make it possible for several different keys to decrypt a message if they satisfy a policy, completely or partially. This allows us to encrypt a message for a certain group of users, and later grant new users access to it without having to decrypt it. A key management service (KMS) issuing keys based on some characteristics, which can be used to decrypt message encrypted previously. The term "Functional Encryption" has come to describe a wide spectrum of modern cryptography techniques, including identity-based encryption and attribute-based encryption, which will be described and demonstrated in the following sections.

2.4.1 Identity-based encryption

Public key systems rely on a trusted CA, issuing certificates assuring the binding between a public key and an claimed owner. The user generate their key pair themselves, then the public key has to be signed by a trusted certificate authority. Now the public key can be verified, assuring that nobody are forging it. Each user have to keep a large archive of keys belonging to whom he wants to communicate. More problems arise when a user wants to declare his key invalid and revoke it. All this makes for a big infrastructure of CAs and revocation mechanisms. IBE[4] introduce an approach where a users id act as the public key, this identity can typically be an e-mail address or a user name. There are no CAs, but instead each domain have a KMS with a master public/secret key pair. The master public key can be used in conjunction with the id of the desired receiver to encrypt the message. The receiver can then decrypt the cipher text using his private key. This private key is extracted by the KMS from the identity of the specific user. The removal of the CAs introduces another problem, since the users no longer generate their own key pair, a lot power is now in the hands of the KMS. Algorithms using a KMS have to take into consideration that this service have complete control over all keys, and can in fact generate any key. The life cycle of a system implementing IBE consist of 4 algorithms.

– Setup - Taking some security parameters 1^k a master public/secret key pair (mpk, msk) are generated.

– Key delegation - Using mpk, msk and some id generating a sk_{id} for the specific user.

– Encryption - Encrypts a message m using mpk and the id of the desired receiver.

– Decryption - Decrypts cipher text ct using sk_{id}, obtaining m.

The scheme is demonstrated using the Charm framework and the implementation described by Waters[22].

See the following commands:

```
>>> from charm.toolbox.pairinggroup import PairingGroup, GT
>>> from charm.schemes.ibenc.ibenc_waters09 import DSE09
>>> group = PairingGroup('SS512')
>>> ibe = DSE09(group)
>>> mpk, msk = ibe.setup()
>>> ID = "student@stud.ntnu.no"
>>> secret_key = ibe.keygen(mpk, msk, ID)
>>> msg = group.random(GT)
>>> ct = ibe.encrypt(mpk, msg, ID)
>>> decrypted_msg = ibe.decrypt(ct, secret_key)
>>> msg == decrypted_msg
True
>>>
```

Notice that anybody can encrypt a message for any user without having a public key stored locally, we simply use the master public key together with the identity of the recipient. This removes a lot of overhead known from public key infrastructure (PKI), we now only need one key for each domain. IBE is a somewhat more intuitive scheme, since the identity of the recipients is used as the public key, thus no connection between different public keys and user identities have to be stored.

2.4.2 Attribute-based encryption

Attribute-based encryption as explained by Goyal et al. [13] introduce an encryption scheme based on user attributes, from which the secret key is generated. Only a key satisfying the access structure can decrypt the cipher text. This is typically useful in cases where the encryptor does not care about who decrypts as long as they satisfy the correct attributes or a set of them. Each user will have a private key corresponding to his set of attributes, from each domain. When encrypting, a policy is specified, this is typically a access tree where the attributes required are leaf nodes and internal nodes are "AND" or "OR"-gates. Different combinations of attributes may therefore be able to encrypt. We can also use these logical gates to construct threshold requirements, where we require k out of n attributes. The encryptor can in example encrypt a message with the policy

("NTNU" and "5th year" and Telematics dpmt.) or
("Professor" and "Telematics dpmt.") or "admin"

Now a user with either the "admin"-attribute or a set including "NTNU", "5th year" and "Telematics dpmt" would be able to decrypt. A user can thus create access structures allowing his id or some combination of other attributes to decrypt without having the attributes himself. It is worth noticing that ABE is a generalisation of

IBE since an identity can be used as an attribute. thus we have a similar structure of algorithms as in IBE (subsection 2.4.1).

– Setup - Taking some security parameters 1^k a master public/secret key pair (mpk, msk) are generated.

– Key generation - Using mpk, msk and some S describing the set of attributes - generates a sk for the specific attribute combination.

– Encryption - Encrypts a message m using mpk and an access structure A describing the policy of the encryption. The attributes in the access structure will define who's able to decrypt.

– Decryption - Decrypts cipher text ct using sk corresponding to an attribute set S, obtaining m.

Charm also include an implemented ABE scheme based on Waters[21], which can be used to describe how the algorithms work. This construction will later be used in the implementation of a prototype attribute-based key exchange application. The implementations are written in python and are thus easily readable. A walkthrough of the ABE protocol using the implementation will be used to demonstrate the algorithms mentioned. The implementation and a sample run of the methods are included for each of the algorithms.

Setup

```
def setup(self):
    g1, g2 = group.random(G1), group.random(G2)
    alpha, a = group.random(), group.random()
    e_gg_alpha = pair(g1,g2) ** alpha
    msk = {'g1^alpha':g1 ** alpha, 'g2^alpha':g2 ** alpha}
    pk = {'g1':g1, 'g2':g2, 'e(gg)^alpha':e_gg_alpha,
    'g1^a':g1 ** a, 'g2^a':g2 ** a}
    return (msk, pk)
```

```
>>> from charm.toolbox.pairinggroup import PairingGroup,ZR,
    G1,G2,GT,pair
>>> from charm.schemes.abenc.abenc_waters09 import CPabe09
>>> groupObj = PairingGroup('SS512')
>>> cpabe = CPabe09(groupObj)
>>> (msk, mpk) = cpabe.setup()
>>> print msk
```

```
    {'g1^alpha': <pairing.Element object at 0x7fc30b633ed0>,
    'g2^alpha': <pairing.Element object at 0x7fc30b606b28>}
>>> print pk
    {'g1^a': <pairing.Element object at 0x7fc30b606b70>,
    'g2^a': <pairing.Element object at 0x7fc30b606bb8>,
    'g2': <pairing.Element object at 0x7fc30b627e40>,
    'g1': <pairing.Element object at 0x7fc30b627d68>,
    'e(gg)^alpha': <pairing.Element object at 0x7fc30b633f18>}
>>>
```

This class can be used as shown in the following demonstration, the environment from which the methods are run already have defined a elliptic curve with bilinear mapping which can be used. The master public and private key pair is then store locally at a server acting as a KMS. After initialising the protocol we can generate a secret key using a defined set of attributes. For this we have the key generation method as following. This will be used by the KMS to generate secret keys for users in the protocol. How these keys are distributed is a separate concern which is not dealt with in this report. It is demonstrated how the keys are generated but now how they are distributed to the correct users. It is assumed that each user can receive their key securely from the KMS.

Key Generation

```python
def keygen(self, mpk, msk, attributes):
    t = group.random()
    K = msk['g2^alpha'] * (pk['g2^a'] ** t)
    L = pk['g2'] ** t
    k_x = [group.hash(unicode(s), G1) ** t for s in attributes]

    K_x = {}
    for i in range(0, len(k_x)):
        K_x[ attributes[i] ] = k_x[i]

    attributes = [unicode(a) for a in attributes]

    key = { 'K':K, 'L':L, 'K_x':K_x, 'attributes':attributes }
    return key
```

```
>>> attr_list = ['THREE', 'ONE', 'TWO']
>>> secret_key = cpabe.keygen(pk, msk, attr_list)
>>> print secret_key
    {'K_x': {'TWO': <pairing.Element object at 0x7fc30b606e40>,
```

```
                'THREE': <pairing.Element object at 0x7fc30b606db0>,
                'ONE': <pairing.Element object at 0x7fc30b606df8>},
        'K': <pairing.Element object at 0x7fc30hen during b606d20>,
        'L': <pairing.Element object at 0x7fc30b606c00>,
        'attributes': [u'THREE', u'ONE', u'TWO']}
>>>
```

The secret key includes a list of all the attributes with a corresponding hash value raised to the power of a random value $t \in \mathbb{Z}_p$. Additionally a list of the unicode representations of the attributes are added - this will later be used when decrypting, to check if a given key comply with the policy given in the cipher text. This way we avoid actually trying to decrypt if the key doesn't contain the correct attributes. The public parameters in *pk* must be published together with the secret keys of each user. We can now generate a message which can be encrypted. A major problem when doing ABE is preventing collusion attacks, where a group of users try to combine their attributes trying to satisfy a more restrictive access structure then what their individual sets of attributes allow. This construction avoids this by randomising each secret with a generated exponent t. When decrypting, each share is raised to the power of this t, which is supposed to bind the components of each key together. During decryption these shares are only relevant to the particular key used in that exact run of the decryption algorithm.

Encryption

```python
def encrypt(self, pk, M, policy_str):
    # Extract the attributes as a list
    policy = util.createPolicy(policy_str)
    p_list = util.getAttributeList(policy)
    s = group.random()
    C_tilde = (pk['e(gg)^alpha'] ** s) * M
    C_0 = pk['g1'] ** s
    C, D = {}, {}
    secret = s
    shares = util.calculateSharesList(secret, policy)

    # ciphertext
    for i in range(len(p_list)):
        r = group.random()
        if shares[i][0] == p_list[i]:
           attr = shares[i][0].getAttribute()
           C[ p_list[i] ] = ((pk['g1^a'] ** shares[i][1]) *
           (group.hash(attr, G1) ** -r))
```

```
        D[ p_list[i] ] = (pk['g2'] ** r)

    return { 'C0':C_0, 'C':C, 'D':D , 'C_tilde':C_tilde,
             'policy':unicode(policy_str), 'attribute':p_list }
```

```
>>> policy = '((ONE or THREE) and (TWO or FOUR))'
>>> msg = group.random(GT)
>>> cipher_text = cpabe.encrypt(master_public_key, msg, policy)
>>> print msg
>>> print cipher_text
    {
    'C': {
            u'TWO': <pairing.Element object at 0x7f0407de4228>,
            u'FOUR': <pairing.Element object at 0x7f0407de42b8>,
            u'THREE': <pairing.Element object at 0x7f0407e5acd8>,
            u'ONE': <pairing.Element object at 0x7f0407e5af18>},
    'D': {
            u'TWO': <pairing.Element object at 0x7f0407e5adb0>,
            u'FOUR': <pairing.Element object at 0x7f0407e5afa8>,
            u'THREE': <pairing.Element object at 0x7f0407e5ae88>,
            u'ONE': <pairing.Element object at 0x7f0407e5af60>},
    'attribute': [u'ONE', u'THREE', u'TWO', u'FOUR'],
    'C_tilde': <pairing.Element object at 0x7f0407e5ad68>,
    'policy': u'((ONE or THREE) and (TWO or FOUR))',
    'C0': <pairing.Element object at 0x7f0407e5ac90>}
```

Before encrypting, a policy is specified, this will be the access structure used in the encryption. Since the protocol relies on pairings, only pairing elements can be used, a random message m is thus generated from the group to be used in the demonstration. If we were to encrypt some kind of readable message we would need an adapter on top, mapping messages to pairing elements. This paper will focus on applications where this is not needed - random group elements is sufficient for the constructions presented later. The encryption method starts off by extracting the attributes from the policy provided, then a random group object is generated and used together with the public key and the message to calculate a internal cipher text. Using this secret a set of shares are generated according to the LSSS in 2.2.1. These shares can now be used to recover s when decrypting.

Decryption

```
def decrypt(self, pk, sk, ct):
    policy = util.createPolicy(ct['policy'])
```

```
    pruned = util.prune(policy, sk['attributes'])
    if pruned == False:
        return False
    coeffs = util.getCoefficients(policy)
    numerator = pair(ct['C0'], sk['K'])

    # create list for attributes in order...
    k_x, w_i = {}, {}
    for i in pruned:
        j = i.getAttributeAndIndex()
        k = i.getAttribute()
        k_x[ j ] = sk['K_x'][k]
        w_i[ j ] = coeffs[j]

    C, D = ct['C'], ct['D']
    denominator = 1
    for i in pruned:
        j = i.getAttributeAndIndex()
        denominator *= ( pair(C[j] ** w_i[j], sk['L']) *
        pair(k_x[j] ** w_i[j], D[j]) )
    return ct['C_tilde'] / (numerator / denominator)
```

```
>>> decrypted = cpabe.decrypt(master_public_key, secret_key, cipher_text)
>>> decrypted == msg
True
>>>
```

Decryption is done using the public parameters and a secret key corresponding to a set of attributes. First step in the decryption is to compare the access structure and the attributes present in the secret key. If the policy is not fulfilled the method can return straight away. The pruned method performs this validation and returns a "pruned" list of attributes. This is the minimal subset of the attributes satisfying the policy - in example a set including both childes of a "OR" node would be pruned to only include one of these. Finally the secrets are combined and used to recover the message.

could show all the math describing how M are recovered (?)

From the scheme described it is noticeable from the encryption method that anybody can in fact encrypt for any set of attributes, as long as they have the master public key. The authentication is not mutual, the encryptor doesn't have to have any specific attributes to be able to encrypt. The protocol only provide assurances that nobody without the correct attribute set can decrypt the message, this is sufficient when used as a public key encryption mechanism, but might not hold in cases where mutual authentication is required.

2.5 Key Exchange

A fundamental requirement in many cryptographic schemes is a way of establishing a common secret to be used later to achieve confidentiality or integrity. This is usually solved using a key exchange scheme. In many scenarios we may as well have a group of players, in example conference calls where the participants are known in advance. A different scenario occurs when we don't know how many and who the participants is, and they may join and leave dynamically.

2.5.1 Group Diffie-Hellman Key Exchange

The Diffie-Hellmann key exchange algorithm in its original form allows two parties to jointly establish a common secret key which later can be used to encrypt traffic and etc. Since the introduction of the 2-party Diffie-Hellmann researchers tried to extend it to support groups of parties [19, 9]. These configurations allow several parties, already in some sort of ring or group, typically a multicast group or similar network, to establish a common session key. In the 2-party Diffie-Hellman a cyclic group G = $\langle x \rangle$ of prime order p is chosen carefully. Then each party chose a random number, a and b, then g^a and g^b can be exchanged and the common secret key g^{ab} computed. The group configuration of the scheme uses the same principle only with several participants as shown in Figure 2.1. The configuration is the same for n players. The scheme starts of with the first player raising g to the power of his private key and sends this value to the next player in the chain. He then raises the received value to the power of his private key and sends it, plus the intermediate values on to the next player, this continues until the last player receives the set, he can now compute the session key $g^{x_1,x_2 \cdot x_3,\ldots,x_n}$. An attacker would have been able to see all the sent combinations, but none of these combine into the session key. New players can not easily join or leave since all previous players would have to update their set. The group Diffie-Hellman can be extended further to a more dynamic variant, allowing joining and leaving within the group of players[7].

2.6 Hybrid Public key Encryption

A hybrid encryption scheme [16] consists of a public key encryption technique and a symmetric key encryption technique, from which the former, key encapsulation mechanism (KEM), is used to encrypt some key K, and the latter, data encapsulation mechanism (DEM), encrypts the data. This setup can be applied using a variety of different cryptographic systems for both the KEM and DEM.

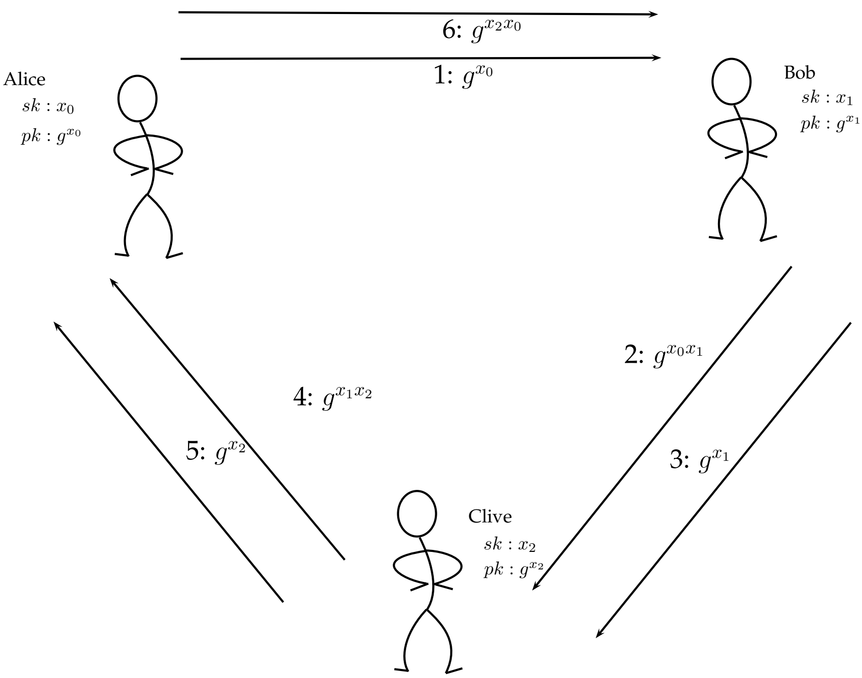

Figure 2.1: D-H group key exchange

2.6.1 Key encapsulation mechanism

KEM [17] is a technique where a random key K is generated together with its encryption C - the encapsulation. This is useful for distribution of symmetric keys that can be used again to generate session keys for two or several parties, depending on the encapsulation mechanism. A KEM consists of three algorithms:

– Key generation - generation of the symmetric key used by the DEM.

– Encryption - used to encrypt the generated key, usually using some public key.

– Decryption - reveals the symmetric key from the encapsulation.

This configuration will be used when constructing key exchange schemes using both IBE and ABE. Hybrid encryption schemes are basically doing key exchange. The focus will be on the setting with several users doing key exchange, a sollution is then to use a hybrid scheme to exchange encapsulations of randomly generated symmetric keys which can be combined to one sessions key. This session key can then be used by the DEM to communicate securly.

Chapter 3

Functional Key Exchange

When communicating on the Internet it is important to control what entities have access to the messages. In most cases it is important that the users can trust that the their communication cannot be stolen or eavesdropped on. Encryption is used to secure communication, to do this efficiently a shared key is usually needed. Functional key exchange is in our context defined as a set of key exchange mechanisms using some function to decided if a participant should be allowed to take part in, or be allowed access to, the key exchange. The functions will use some arguments as input and based on these decide if the session key should be output or not. This chapter will explain some proposed schemes trying to adapt this idea, then further explore possibly useful application areas and ideas. Identity-based authenticated key exchange (IB-AKE) and attributed-based authenticated key exchange (AB-AKE) will be used as examples, with most focus on the latter.

3.1 Identity-based authenticated key exchange

IBE as described in 2.4.1, can by utilised to provide two-party mutually authenticated key exchange (AKE) [10]. The approach is based on a Diffie-Hellman key exchange using an elliptic curve. Each party chose random points a, b. a^p, b^p are then encrypted using the other parties public key and then exchanged in succession. B will include p^a which he recieved from a, this is done so that A can verify that B actually was able to decrypt what he sent. B actually adds to what he receives from A by decrypting and adding his contribution and then encrypting again. After decrypting, the session key is the product a^{bp}, which both can calculate. After exchanging secrets, A has to authenticate himself in the same way as B did, by sending the secret he got from B back, to show that he was able to decrypt what B sent. This technique provides mutual implicit authentication between the participants, since only the users with the correct identity can decrypt. Both parties can thus be sure that no other user than the user in possession of the private key corresponding to the identity can produce the secret key. The procedure is shown in Protocol 1.

Protocol 1.

A - given curve and point p		B - given curve and point p
chose random point a		
	$\xrightarrow{IBEnc_B(p^a)}$	
		chose random point b
	$\xleftarrow{IBEnc_B(p^a, p^b)}$	
verify p^a after decrypting using private key		
	$\xrightarrow{IBEnc_B(p^b)}$	
		verify p^b to authenticate that A actually decrypted the message
	sk = p^{ab} sid=(p^a, p^b)	

This implementation demonstrates a scheme for key exchange between two parties with the focus on assuring authenticity of the identities of the participants. This is mostly a more effective implementation of public key crypto systems, the main difference from previously popular systems is the removal of PKI by switching from CAs to KMSs. Point being that the main idea is still to encrypt some message or symmetric key for *one* specific user.

3.2 Attributed-based authenticated key exchange

[12] introduced the concept of AB-AKE using a attribute-based key encapsulation mechanism. In short this is a KEM with ABE as the encryption mechanism.

IN PROGRESS

3.3 Applications

Chapter 4
Design and Implementation

This chapter suggests a prototype implementation of one possible application, on background of the principles and ideas in the previous two sections. The protocol structure will be displayed, together with examples from the code and test runs using the system.

4.1 System specifications

4.2 Models and construction

4.3 Implementation

4.4 System demonstration

Chapter 5

Discussion

References

[1]

[2] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *J. Cryptographic Engineering*, 3(2):111–128, 2013.

[3] James Birkett and Douglas Stebila. Predicate-based key exchange. In *Information Security and Privacy*, pages 282–299. Springer, 2010.

[4] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[5] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer Berlin Heidelberg, 2011.

[6] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: a new vision for public-key cryptography. *Communications of the ACM*, 55(11):56–64, 2012.

[7] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. In *Advances in Cryptology The Dynamic Case*, pages 290–309. Springer Berlin Heidelberg, 2001.

[8] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Dynamic group Diffie-Hellman key exchange under standard assumptions. In LarsR. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 321–336. Springer Berlin Heidelberg, 2002.

[9] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group Diffie-Hellman key exchange. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 255–264. ACM, 2001.

[10] Violeta Cakulev and Ganapathy Sundaram. Ibake: Identity-based authenticated key exchange. 2012.

[11] Ratna Dutta, Rana Barua, and Palash Sarkar. Pairing-based cryptography: A survey. *Cryptology Research Group, Stat-Math and Applied Statistics Unit*, 203, 2004.

[12] M. Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. Attribute-based authenticated key exchange. In *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, pages 300–317, 2010.

[13] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[14] L. Donnerhacke IKS GmbH H. Finney D. Shaw R. Thayer J. Callas, PGP Corporation. Openpgp message format, 11 2007. RFC 4880.

[15] Werner Koch et al. The GNU privacy guard. *Computer software. Available from http://www.gnupg.org*, 2003.

[16] Kaoru Kurosawa. Hybrid encryption. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 570–572. 2011.

[17] Kaoru Kurosawa and Le Trieu Phong. Kurosawa-Desmedt key encapsulation mechanism, revisited. In *Progress in Cryptology - AFRICACRYPT 2014 - 7th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 28-30, 2014. Proceedings*, pages 51–68, 2014.

[18] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613.

[19] Michael Steiner, Gene Tsudik, and Michael Waidner. Diffie-Hellman key distribution extended to group communication. In *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 31–37. ACM, 1996.

[20] Hao Wang, Qiuliang Xu, Han Jiang, and Rui Li. Attribute-based authenticated key exchange protocol with general relations. In *Seventh International Conference on Computational Intelligence and Security, CIS 2011, Sanya, Hainan, China, December 3-4, 2011*, pages 900–904, 2011.

[21] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. *IACR Cryptology ePrint Archive*, 2008:290, 2008.

[22] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, pages 619–636, 2009.