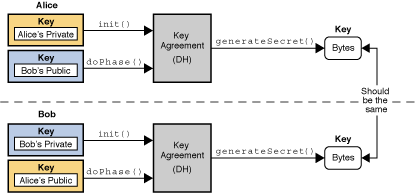
**The KeyAgreement Class**

Key agreement is a protocol by which 2 or more parties can establish the same cryptographic keys, without having to exchange any secret information.



[Description of Figure 14: The KeyAgreement Class](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec_image_descriptions.html" \l "keyagreement)

Each party initializes their key agreement object with their private key, and then enters the public keys for each party that will participate in the communication. In most cases, there are just two parties, but algorithms such as Diffie-Hellman allow for multiple parties (3 or more) to participate. When all the public keys have been entered, each KeyAgreement object will generate (agree upon) the same key.

The KeyAgreement class provides the functionality of a key agreement protocol. The keys involved in establishing a shared secret are created by one of the key generators (KeyPairGenerator or KeyGenerator), a KeyFactory, or as a result from an intermediate phase of the key agreement protocol.

**Creating a KeyAgreement Object**

Each party involved in the key agreement has to create a KeyAgreement object. KeyAgreement objects are obtained by using one of the KeyAgreement [getInstance() static factory methods](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html" \l "ProviderImplReq).

**Initializing a KeyAgreement Object**

You initialize a KeyAgreement object with your private information. In the case of Diffie-Hellman, you initialize it with your Diffie-Hellman private key. Additional initialization information may contain a source of randomness and/or a set of algorithm parameters. Note that if the requested key agreement algorithm requires the specification of algorithm parameters, and only a key, but no parameters are provided to initialize the KeyAgreement object, the key must contain the required algorithm parameters. (For example, the Diffie-Hellman algorithm uses a prime modulus p and a base generator g as its parameters.)

To initialize a KeyAgreement object, call one of its init methods:

public void init(Key key);

public void init(Key key, SecureRandom random);

public void init(Key key, AlgorithmParameterSpec params);

public void init(Key key, AlgorithmParameterSpec params,

SecureRandom random);

**Executing a KeyAgreement Phase**

Every key agreement protocol consists of a number of phases that need to be executed by each party involved in the key agreement.

To execute the next phase in the key agreement, call the doPhase method:

public Key doPhase(Key key, boolean lastPhase);

The key parameter contains the key to be processed by that phase. In most cases, this is the public key of one of the other parties involved in the key agreement, or an intermediate key that was generated by a previous phase. doPhase may return an intermediate key that you may have to send to the other parties of this key agreement, so they can process it in a subsequent phase.

The lastPhase parameter specifies whether or not the phase to be executed is the last one in the key agreement: A value of FALSE indicates that this is not the last phase of the key agreement (there are more phases to follow), and a value of TRUE indicates that this is the last phase of the key agreement and the key agreement is completed, i.e., generateSecret can be called next.

In the example of [Diffie-Hellman between two parties](https://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html#AppD) , you call doPhase once, with lastPhase set to TRUE. In the example of Diffie-Hellman between three parties, you call doPhase twice: the first time with lastPhase set to FALSE, the 2nd time with lastPhase set to TRUE.

**Generating the Shared Secret**

After each party has executed all the required key agreement phases, it can compute the shared secret by calling one of the generateSecret methods:

public byte[] generateSecret();

public int generateSecret(byte[] sharedSecret, int offset);

public SecretKey generateSecret(String algorithm);