AsiaBSDCon 2014 Tutorial

Kerberos 認証サービスの設定と運用

佐藤 広生 <hrs@allbsd.org>

東京工業大学 / FreeBSD Project

2014/3/13

2014/3/13 (c) Hiroki Sato

終わるまでに理解できる内容

- 「認証」「認可」とは？
- Kerberos 認証サービスの原理と構造
- KDC の構築 (kdc, kadmind, kpasswdd)
- Kerberos 認証のアプリケーション設定
  - login
  - OpenSSH
  - BIND (GSS-TSIG)
  - Apache (HTTP Basic Auth, SPNEGO)
- 複数のKDCの同期 (hprop, ipropd)

認証と認可

- UNIX の伝統的な認証・認可の仕組み
  - /etc/passwd
  - **hrs:1EhW6tUdUn3q.**:20001:20001::0:0:Hiroki Sato:/home/hrs:/bin/tcsh
  - UID とパスワードハッシュの一致でログイン可否を決定

認証と認可

- 認証 (Authentication)
  - 「本当にその UID の持ち主かどうか」を判断する処理
    - 方法：「本人しか知らない情報」と「UID」のペア情報を知っているかチェック
      - 例：共通鍵の事前共有 (pre-shared key)
        秘密鍵・公開鍵ペアの一致を見る
        二要素認証（情報を増やす）

認証と認可

- 認可 (Authorization)
  - 「その資源を使って良いかどうか」を判断する処理
    - 方法：「使って良い人リスト」（認可データベース）と「UID」の照合

認証と認可

- 違いはどこ？
- 認証と認可の身近な例：パスポート
  - 「パスポートを持っている」＝「本人である」ことの証明
  - 「パスポートを持っている」！＝「出国・入国が保証される」
    - 入出国は、犯罪者リスト等の別の情報と照合する
    - パスポートは、本人であることを証明するだけ

認証と認可

- /etc/passwd の認証・認可
  - 「パスワードハッシュが一致する」＝「UID の持ち主」
  - 認可は？
    - ポイント：/etc/passwd が認可データベースも兼ねている
  - 認証される＝認可される
  - そもそもそんな区別は必要なのか？

認証と認可

- 認証・認可の区別がないと困ること
  - 認可のたびに認証が必要
    - ログインしてから、コマンドを打つたびにパスワードが必要だったら？
- 認証されたこと（＝そのUIDの正当な持ち主であること）を保持しておくことで面倒を回避
  - ログイン：プロセスの所有者がUIDと一致するので、子プロセスは自動的にUIDを引き継ぐ

認証と認可

- 認証・認可の区別がないと困ること
  - ログインプロセスのUIDだけで十分か？
    - リモートシステムに対しては使えない
      - ウェブページを開くたびにパスワードを入れる？
      - メールにアクセスするたびにパスワードを入れる？

認証と認可

- 認証・認可の区別がないと困ること
  - 使う人が同じなら、認証は本来一回で良い。
  - 認可は、使いたい資源ごとに判断される必要がある。
  - 区別がないと、不要な認証を何度もやらなければならない

Kerberos が目指すもの

- 信頼できる「認証」を提供するサービス
  - 政府がパスポートを発行するようなイメージ
    - 持ち主の氏名と国籍を保証する
    - パスポートを持ってて何ができるかは、政府は保証しない

Kerberos が目指すもの

- メリット
  - 認証データベースを一元化できる
    - たくさんのマシンにパスワードハッシュが分散しない
    - マシンをセットアップしたら、Kerberos に対応させてUIDを登録するだけで、パスワードハッシュがなくてもログイン可能
  - セキュリティの向上
- 認証を使い回しできる（シングルサインオン）
  - リモートログインのたびにパスワードを打鍵しなくて良い

SSH じゃダメなの？

- システム管理者で SSH を知らないひとはいない（はず）
- Kerberos なんか要らんのでは？

SSH じゃダメなの？

- 最後に .ssh/authorized_keys を編集したのはいつ？

SSH じゃダメなの？

- 公開鍵認証の欠点：revocation が難しい
  - PKI には CRL (Certification Revocation List) がある。
  - OpenSSH には KRL (Key Revocation lists) と RevokedKeys オプションがある。

```
% ssh-keygen -k -f /etc/ssh/revoke_keys -z 1 /path/pubkey.pub
% ssh-keygen -u —k -f /etc/ssh/revoke_keys -z 2 /path/pubkey.pub
```

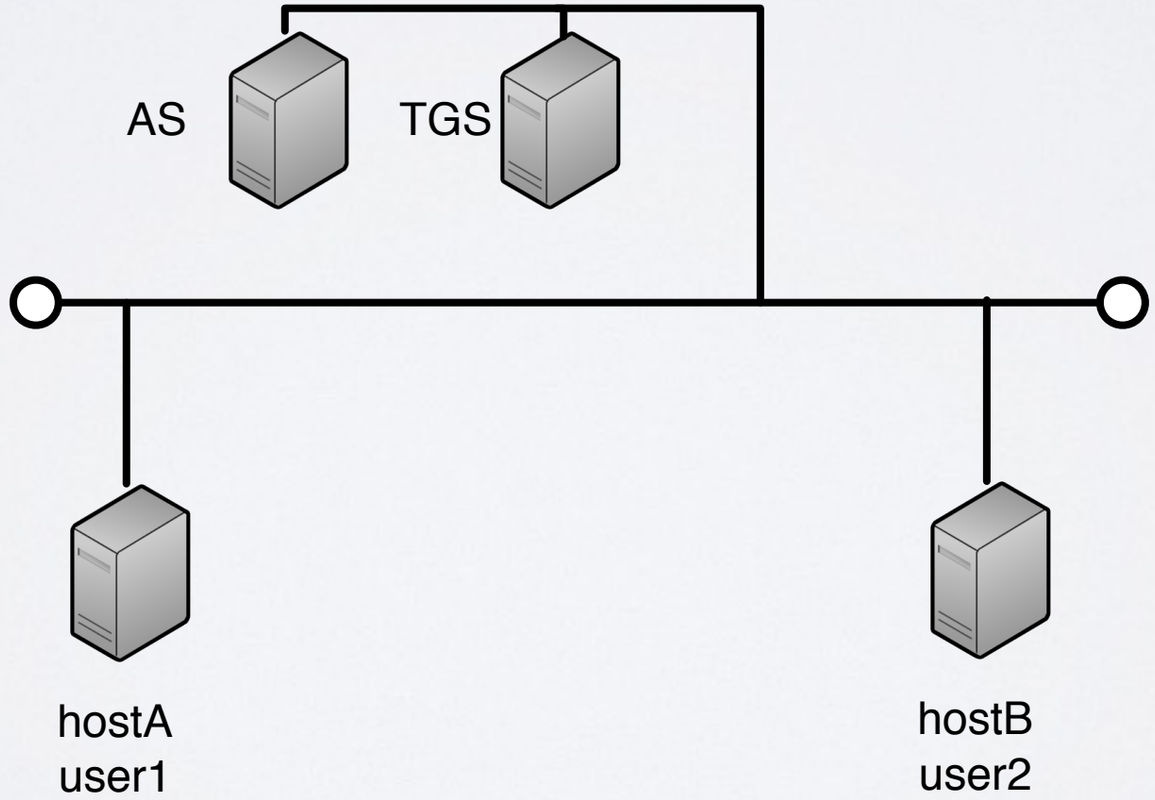
- 多数のホストに authorized_keys ファイルのコピーをつくると、revoke は困難

Kerberos 認証の構造

- 歴史と特徴
  - MIT の Project Athena（実体は分散システムの開発）で 1980 年代に開発されたリモート認証プロトコル
  - 目標：種類の違う数千台規模のマシンの認証を効率的に管理
  - 手段：
    - パスワード情報を一元化（認証サーバ）
    - 認証を再利用できるように（認証チケット）
- 現在普及しているOSのほとんどが Kerberos の技術を利用

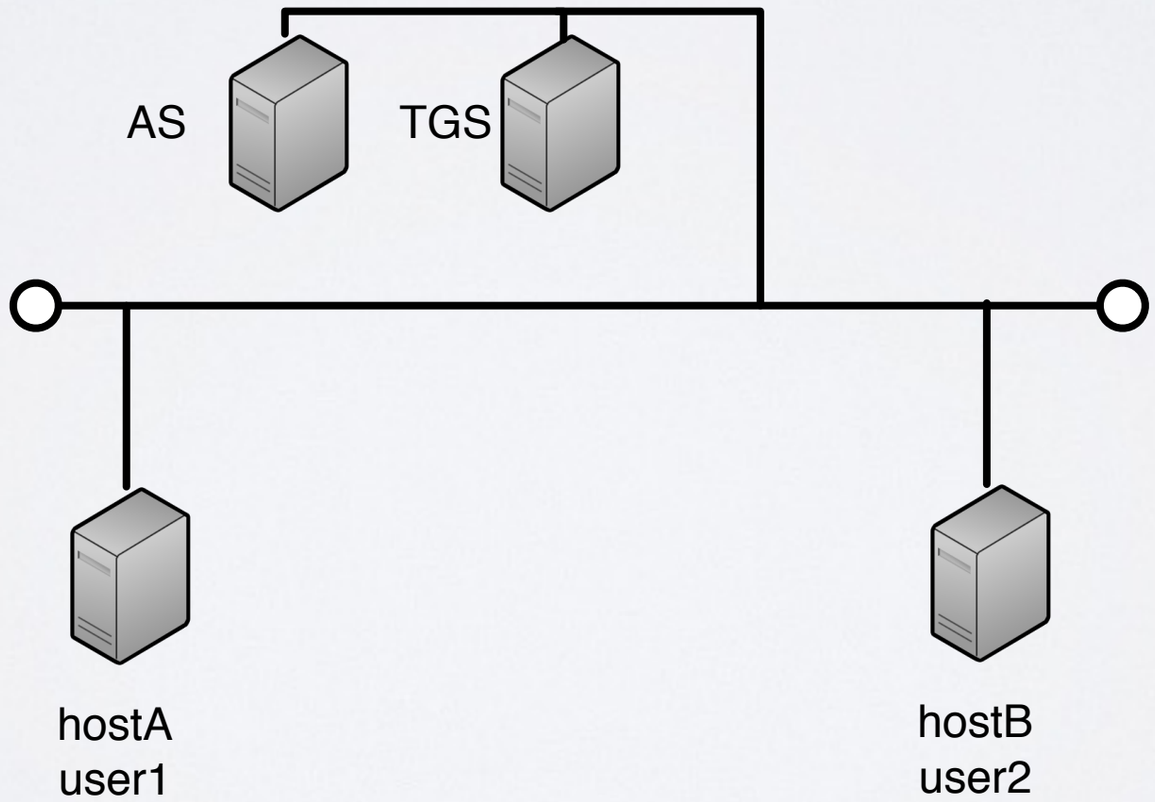
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - 共通鍵暗号を使った、安全な認証システムの構築方法
  - Kerberos 認証の根幹



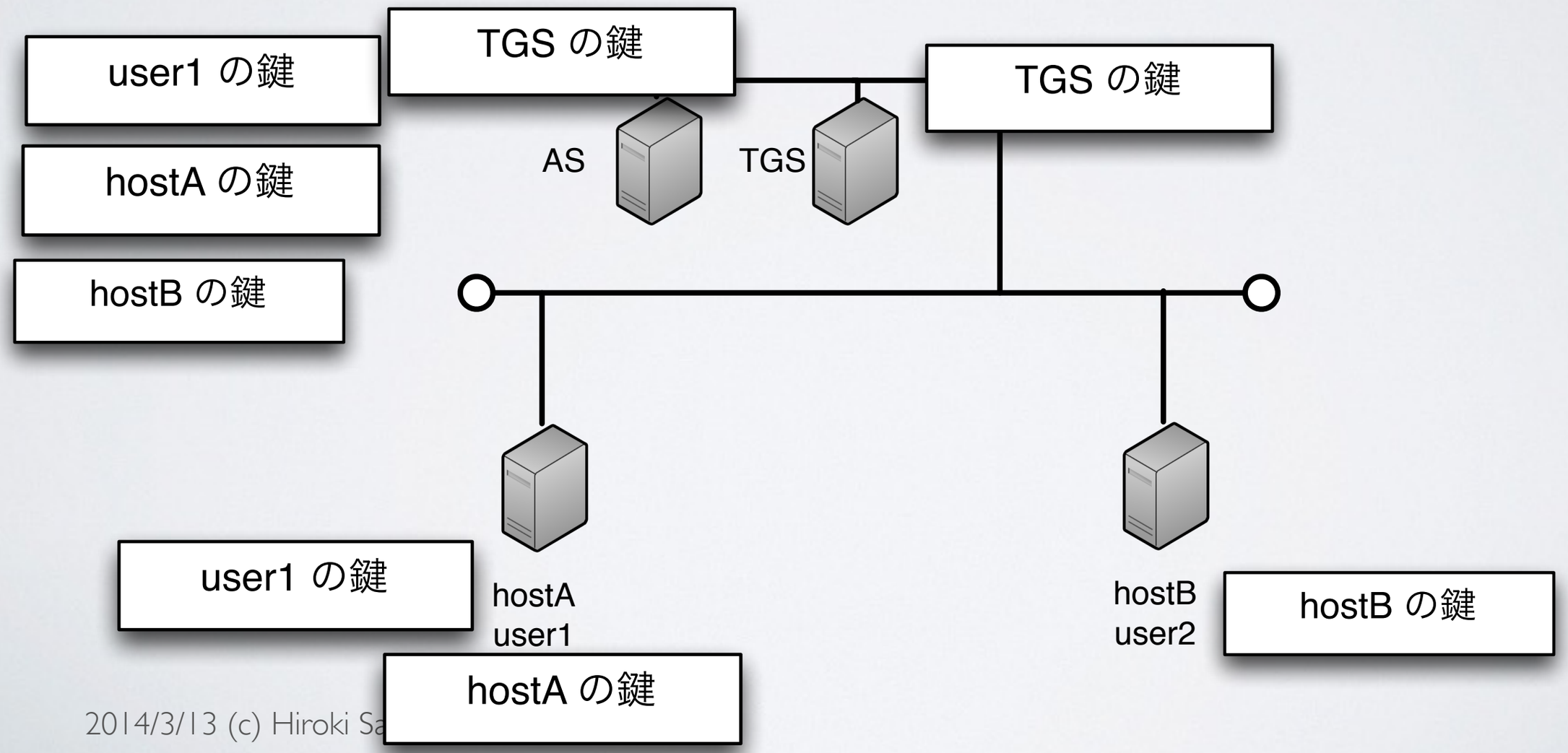
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - シナリオ：hostA の user1 が hostB にログインしたい



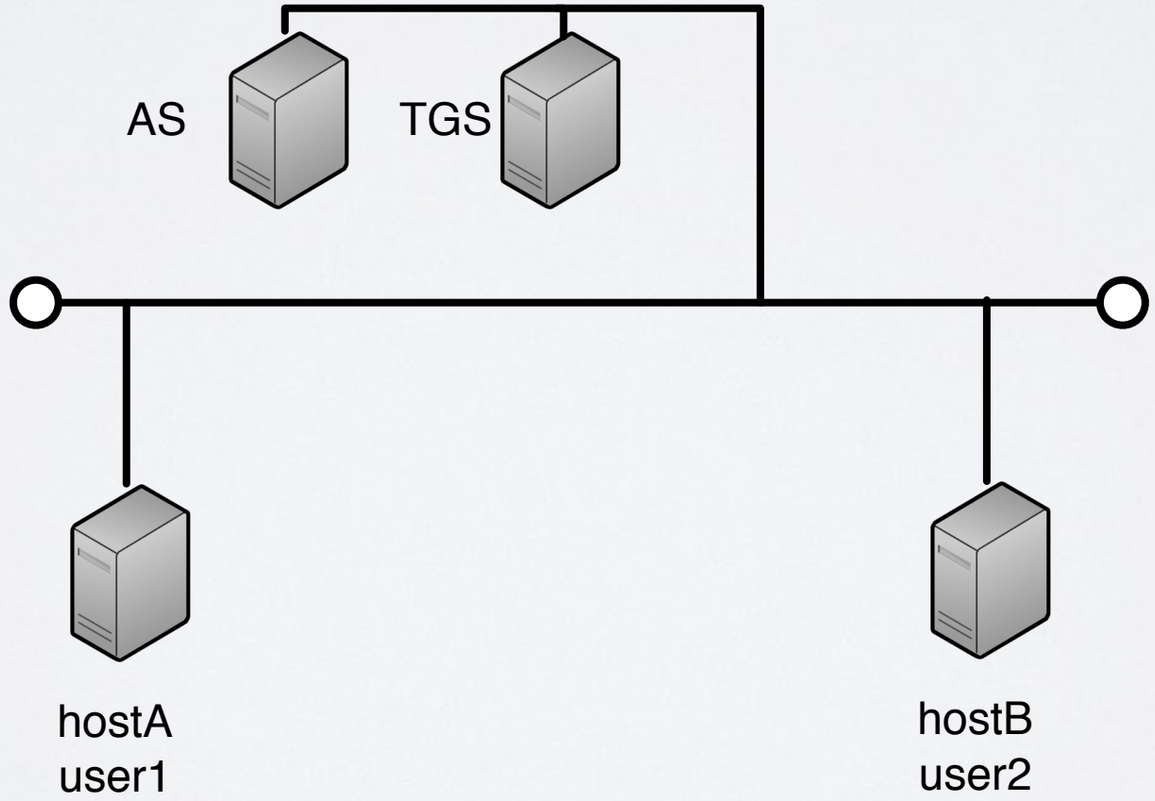
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - 前提：すべての要素は、鍵（パスワード）を持っている
  - 前提：AS（認証サーバ）は、鍵（パスワード）をすべて持っている



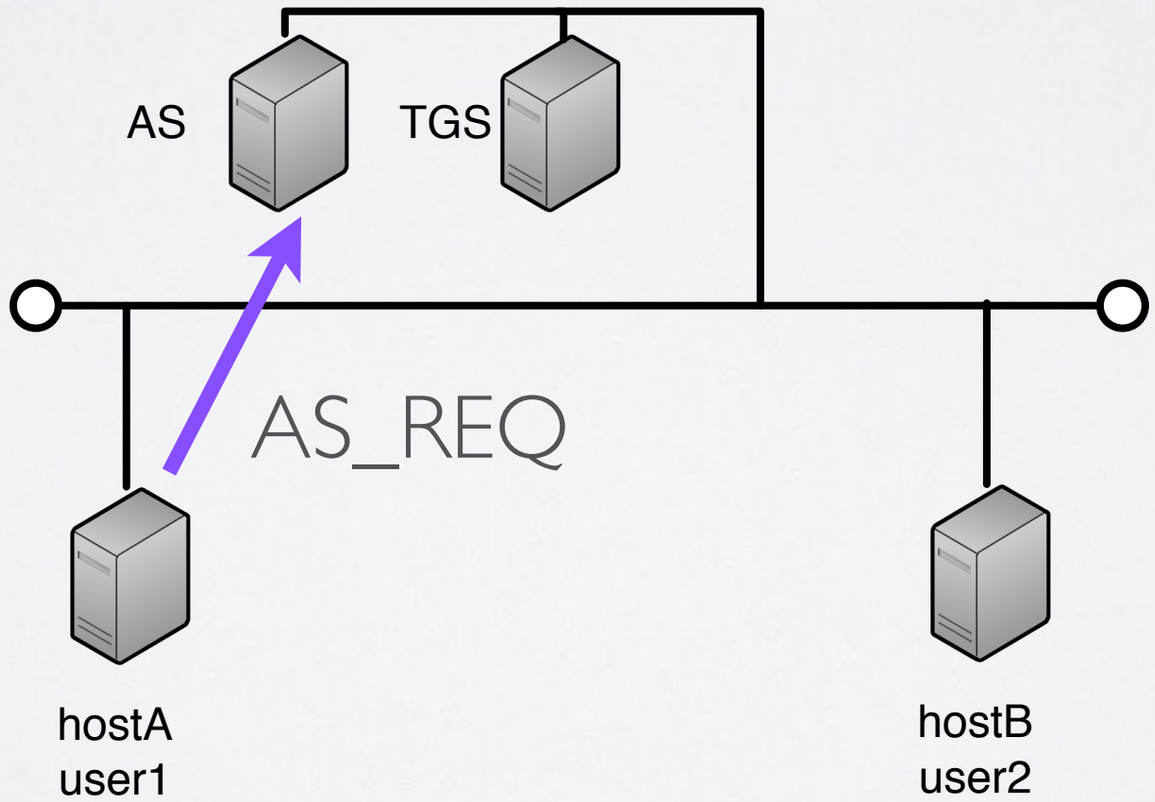
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - なぜか？
  - 事前登録されていることを使って信頼性を構築



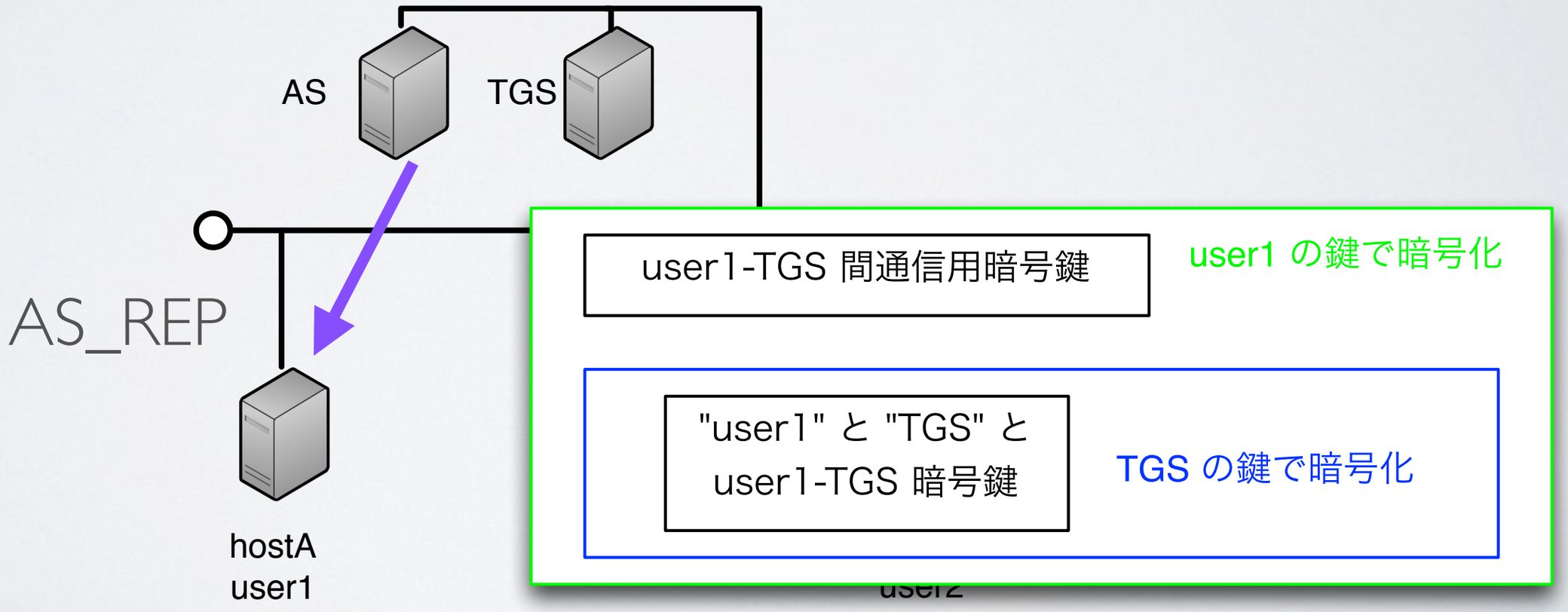
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - ステップ：user1 が AS に認証要求を出す
  - 認証要求は、「名前 “user1” とサービス名 “TGS”」で出す



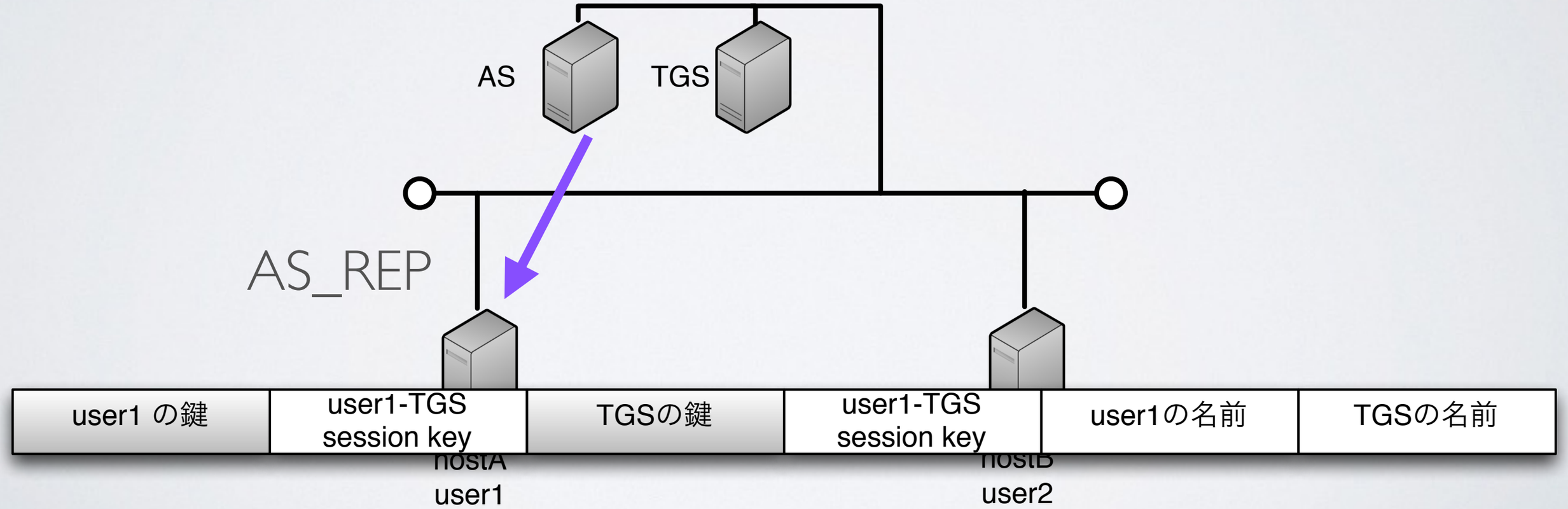
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - ステップ：AS は、user1 に初期チケットを返す
  - 初期チケットは、TGS へのアクセスに使う



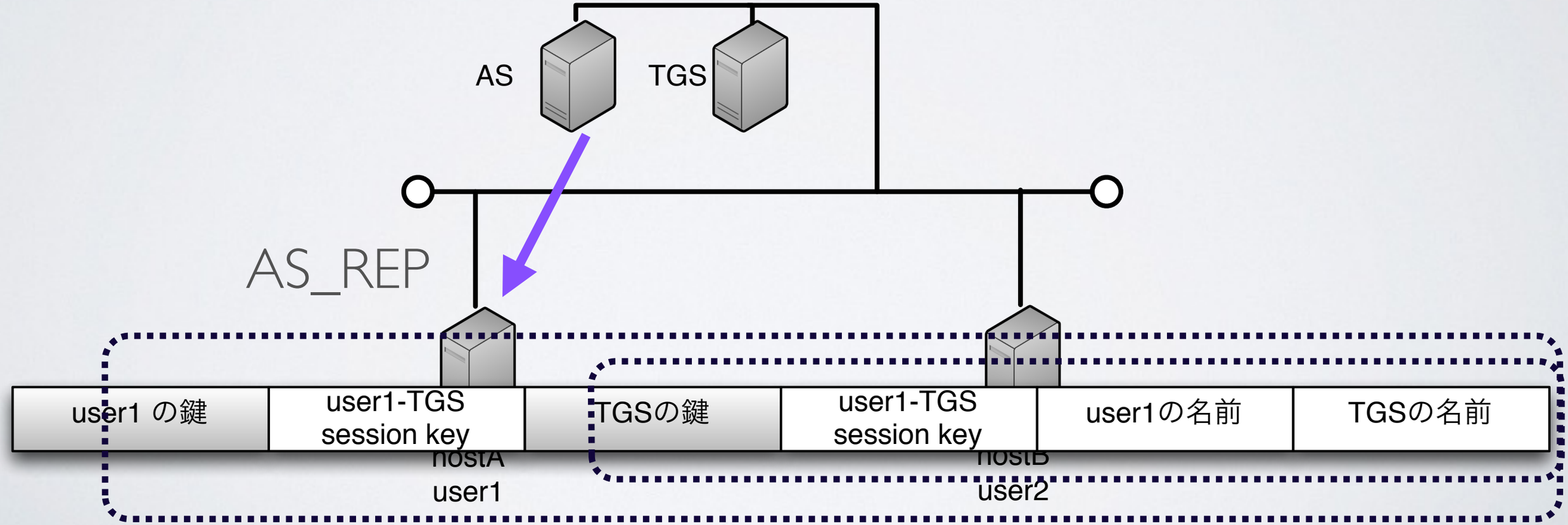
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - ステップ：AS は、userI に初期チケットを返す
  - 初期チケットは、TGS へのアクセスに使う



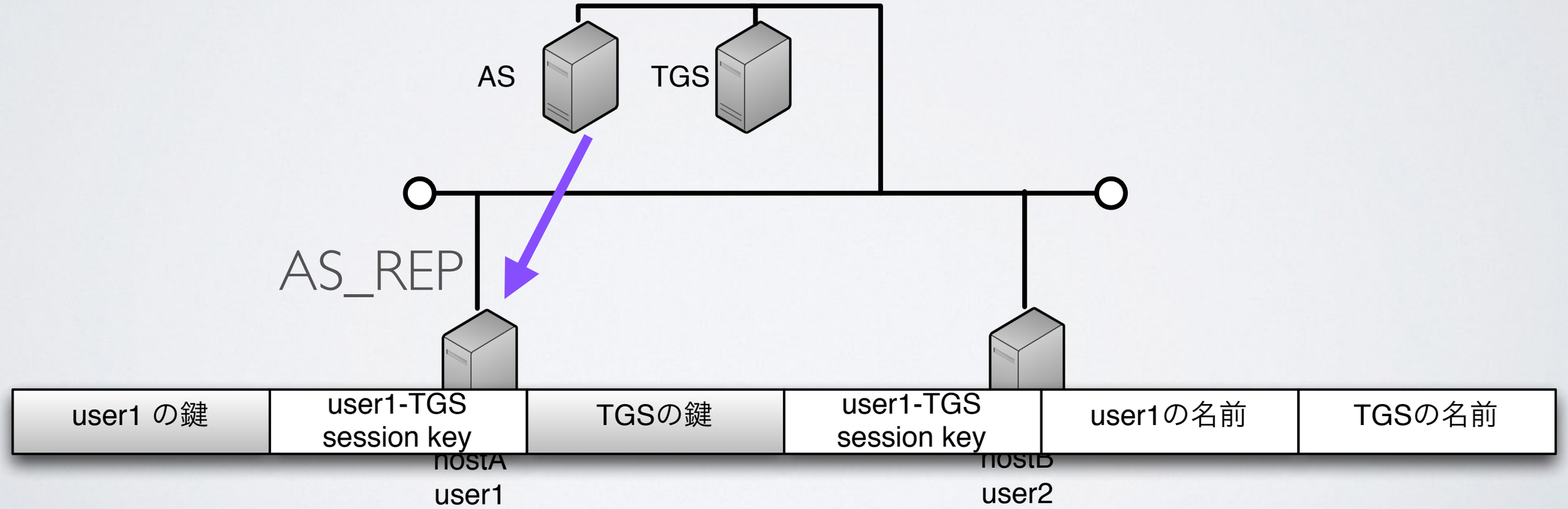
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - ステップ：AS は、userI に初期チケットを返す
  - 初期チケットは、TGS へのアクセスに使う



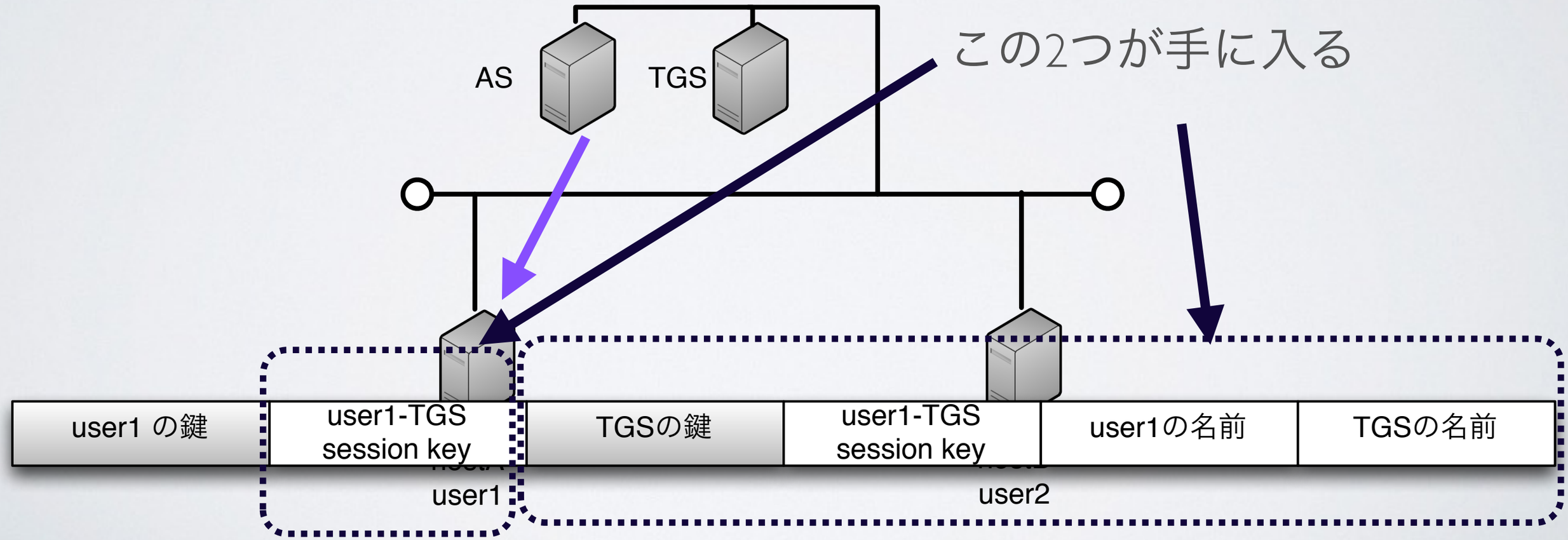
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - userI の鍵（パスワード）で暗号化した部分は、userI なら戻せる
  - TGS の鍵（パスワード）で暗号化した部分は、TGS でしか戻せない



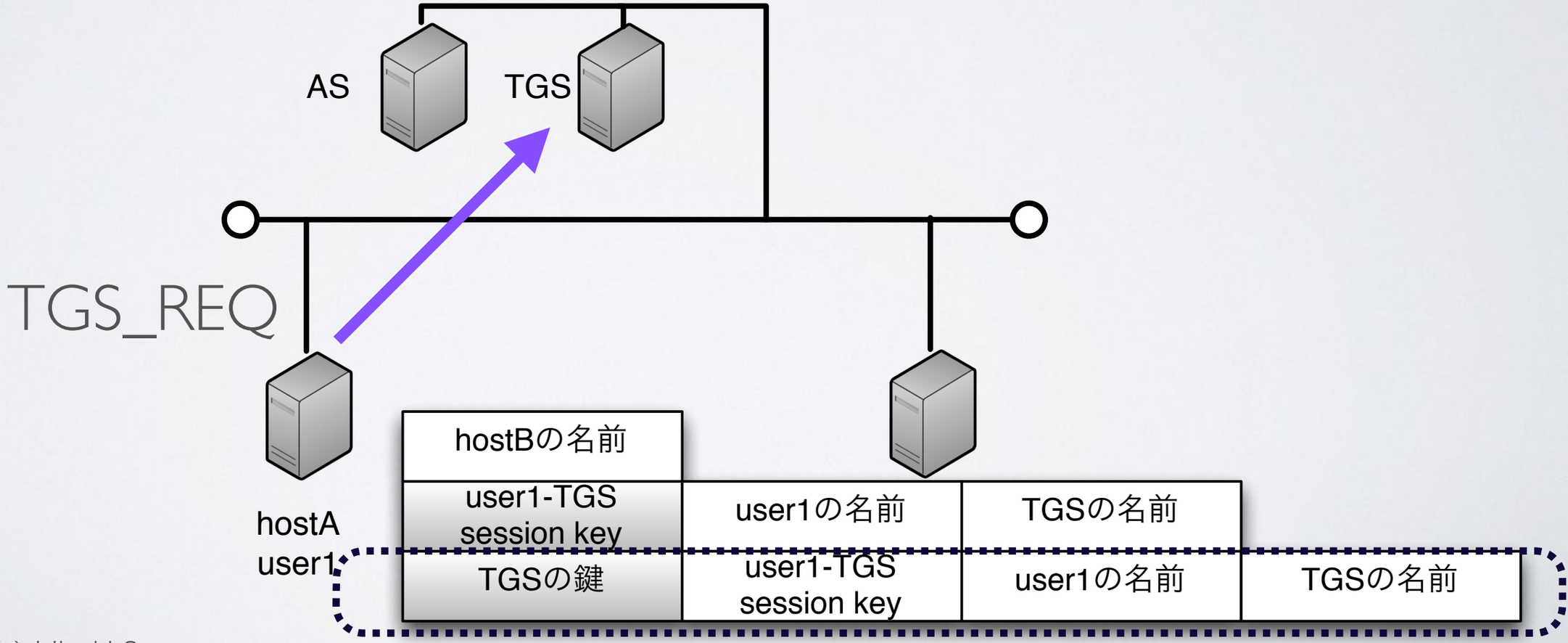
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - userI の鍵（パスワード）で暗号化した部分は、userI なら戻せる
  - TGS の鍵（パスワード）で暗号化した部分は、TGS でしか戻せない



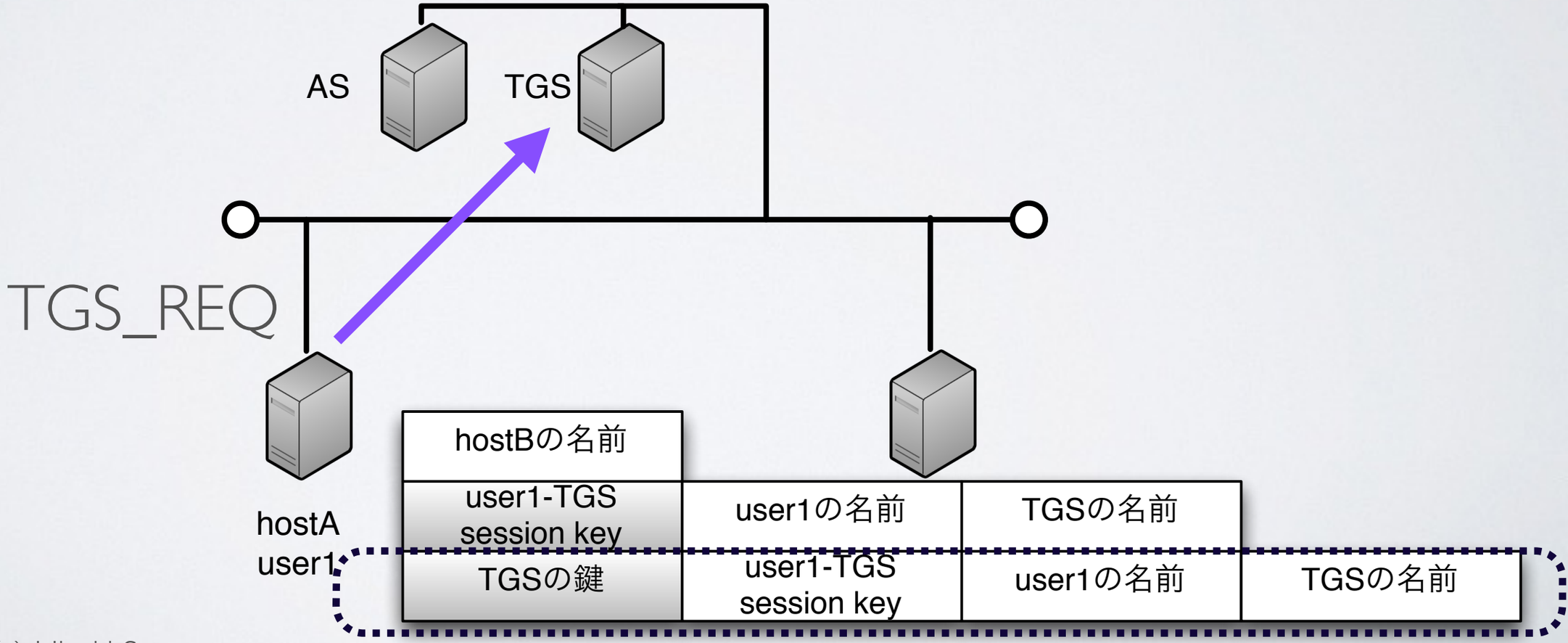
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - user1-TGS 間通信用暗号鍵を使ってTGSと暗号通信開始
  - チケットの残りと、サービス要求をTGS へ送信



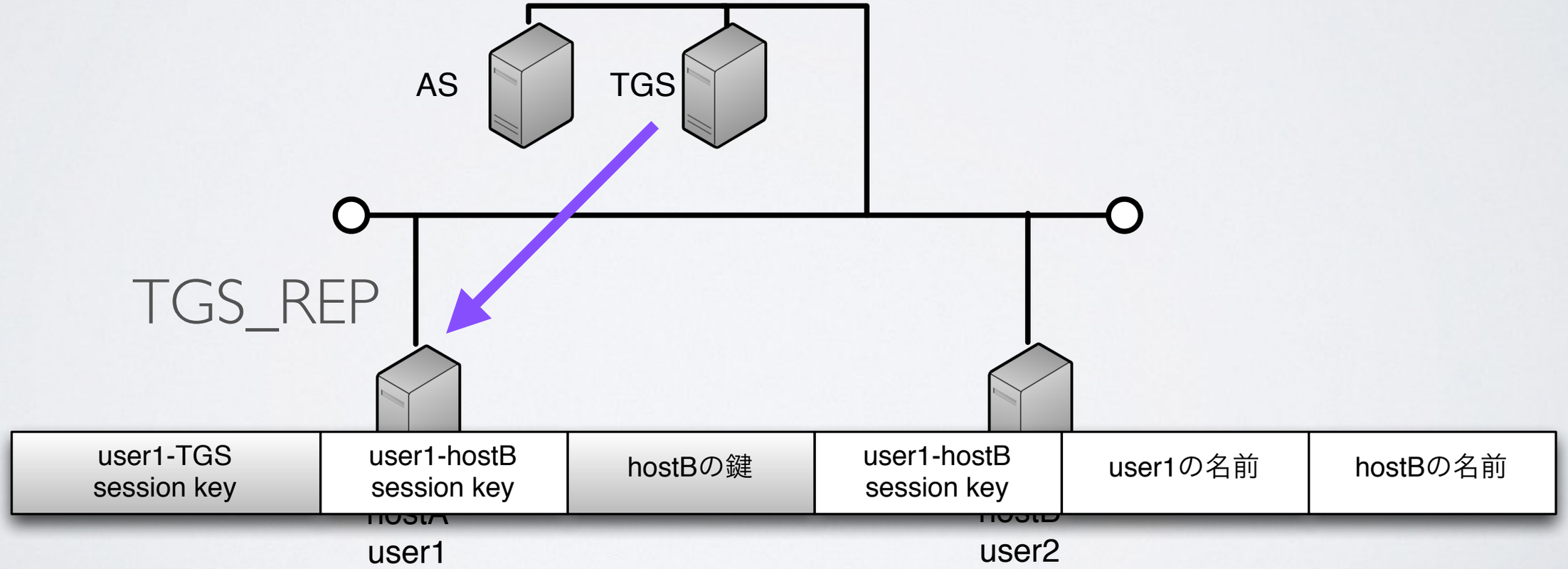
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - user1-TGS 間通信用暗号鍵を使ってTGSと暗号通信開始
  - この時点で、user1 であることが確実に検証できる



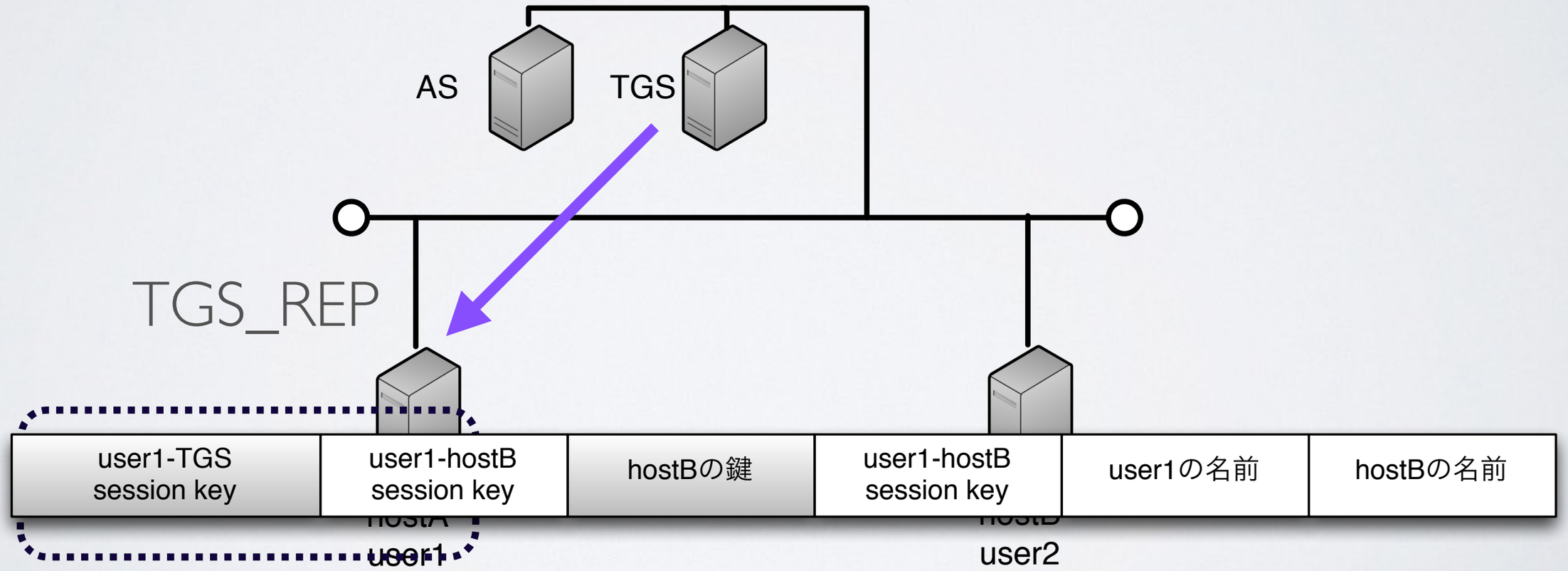
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - user1-TGS 間通信用暗号鍵を使ってTGSと暗号通信開始
  - hostB との通信に使う暗号鍵と、要求内容が入ったデータを受信



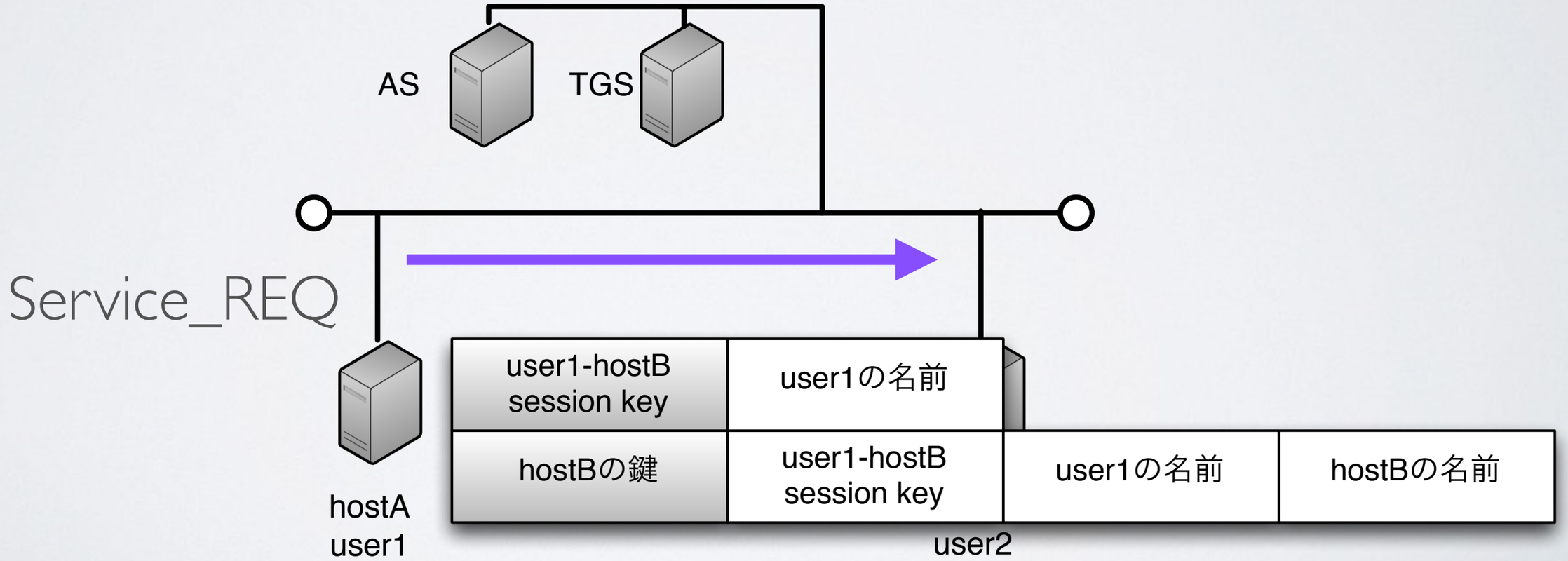
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - これで user1-hostB 間の暗号鍵が手に入った！
  - hostB への通信を開始



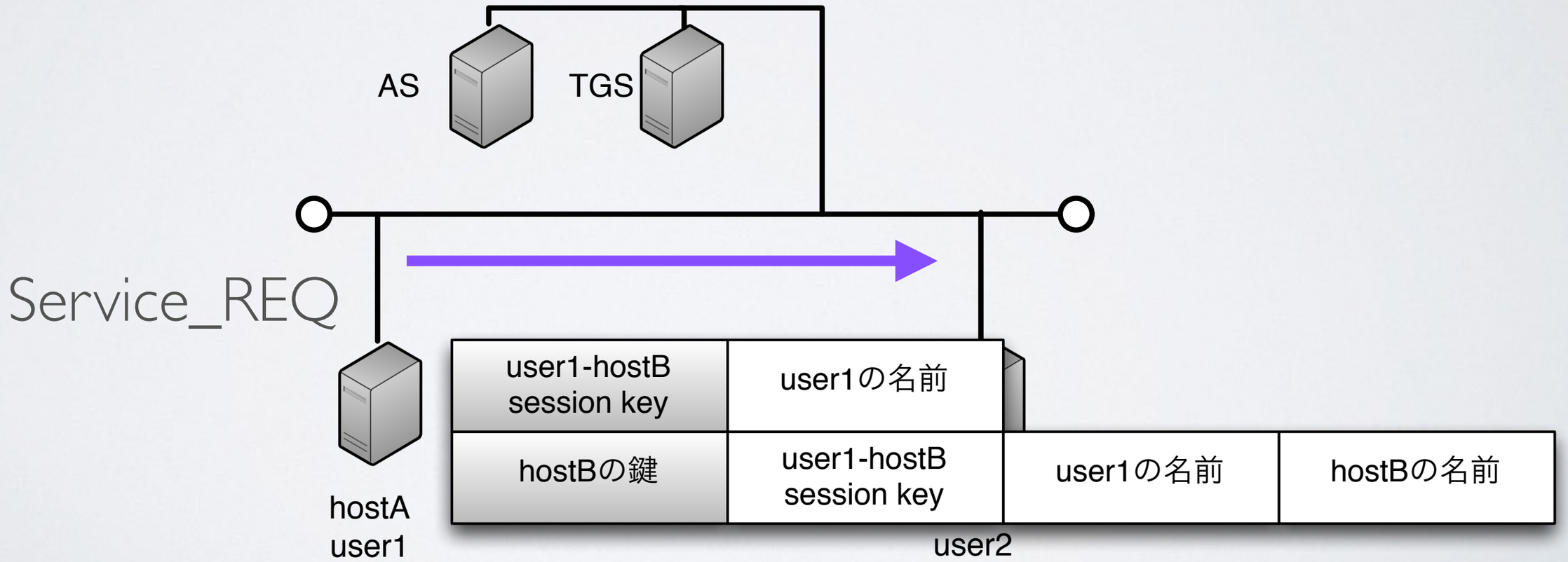
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - user1 であることの証明
  - user1-hostB の通信用の正しい暗号鍵



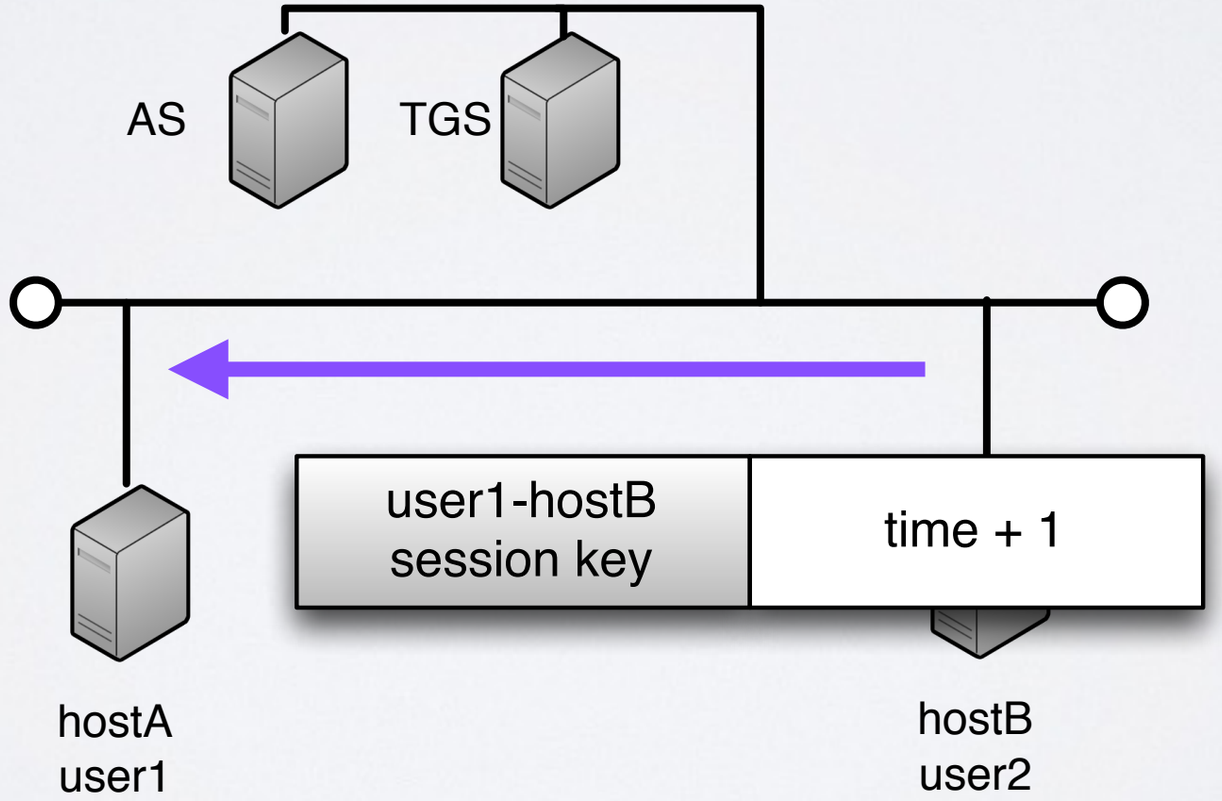
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - hostB は、両方の鍵を持っているので復元できる
  - 復元できる＝認証成功！



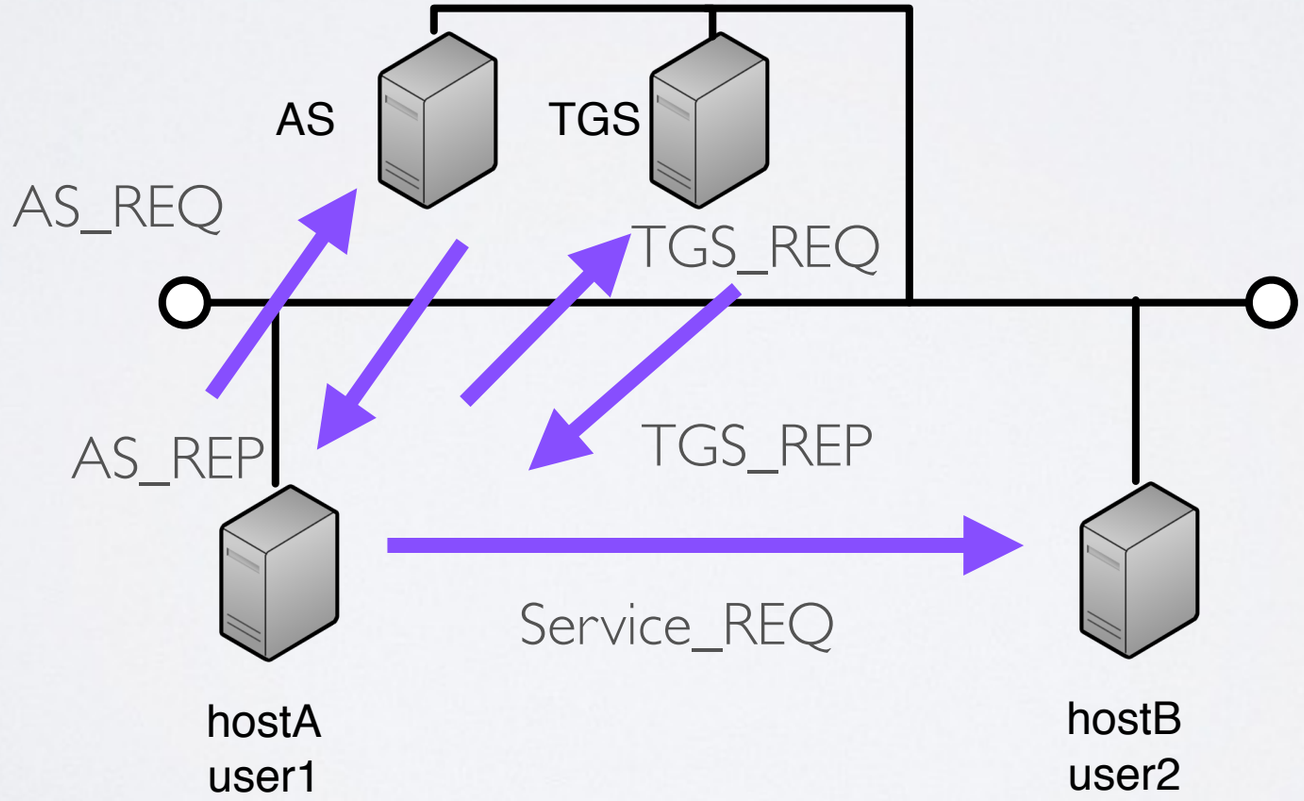
Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - 念のため逆方向にもチェック（タイムスタンプ＋１を送る）



Kerberos 認証の構造

- キーとなる技術: Needham-Schroeder 共通鍵プロトコル
  - まとめ：user1 が hostB にログインするには、5 工程必要
  - サーバ側(hostB)は、ASにもTGSにもアクセスしていない！



Kerberos 認証の構造

- 必要な要素
  - AS：認証データベース
  - TGS：認証を再利用するためのデータ（チケット）を配る
  - チケットをサーバに送るだけで、userIであることを確認できる！

Kerberos 認証の構造

- 実際の実装
  - KDC = AS +TGS（鍵配布センター）
    - 認証データベースを保持して、TGSの機能も持つ
      ネットワークデーモン
  - クライアントは KDC を
    - /etc/krb5.conf にある IP アドレスから探す
    - DNS の SRV RR を見て探す
- Kerberos 対応のソフトウェアでないと、この認証は使われない
  - PAM を使って KDC を使うように変えることは可能

Kerberos 認証の構造

- もう一度復習
  - **認証サーバ・チケット配布サーバ：KDC**
    - 鍵（パスワード）を全部知ってるサーバ
  - **認証クライアント（2者関係）**
    - 主体はユーザ、アプリケーション、など
    - それぞれ鍵（パスワード）を持っている
    - 例：リモートログインの場合
      - 「ユーザ」と「対象ホスト」

5分休憩

・後半は設定です

設定方法

- **KDC の構築**
  - kdc デーモン
  - kadmind デーモン
  - kpasswdd デーモン
- **クライアントの設定**
  - login
  - OpenSSH
  - BIND
  - Apache

KDC の構築

- Kerberos の実装: MIT Kerberos と KTH Heimdal が有名
- FreeBSD は KTH Heimdal がベースに入っている
  - /usr/libexec/kdc: KDC デーモン
  - /var/heimdal: KDC が認証データベースを置くところ
  - /etc/krb5.conf: クライアント用設定ファイル
  - /etc/krb5.keytab: デフォルト鍵ファイル

KDC の構築

- ステップ
  - レルム(realm)を決める
  - マスタ鍵生成
  - 管理用プリンシパルを追加
  - kdc デーモンを起動
  - kadmind デーモンを起動

KDC の構築

- 1 ：レルムを決めて kadmin -l init
  - レルム＝ドメイン名のようなもの（一致させても良い）
  - 全部大文字にするのが普通
- このコマンドで認証データベース/var/heimdal/heimdal.dbが初期化

```
# kadmin -l init ALLBSD.ORG
```

KDC の構築

- 1 ：レルムを決めて kadmin -l init
  - レルム＝ドメイン名のようなもの（一致させても良い）
  - 全部大文字にするのが普通
  - このコマンドで認証データベース/var/heimdal/heimdal.dbが初期化
- 2 ：マスタ鍵を kadmin -l stash で設定

```
# kadmin -l stash
Master key:
Verifying - Master key:
kadmin: writing key to "/var/heimdal/m-key"
```

KDC の構築

- マスタ鍵：/var/heimdal/m-key
- 認証データベース：/var/heimdal/heimdal.db
  - マスタ鍵＝認証データベースの暗号鍵
  - root 以外は読めないように注意すること！

```
# kadmin -l init ALLBSD.ORG
Realm max ticket life [unlimited]:
Realm max renewable ticket life [unlimited]:
# kadmin -l stash
Master key:
Verifying - Master key:
kadmin: writing key to "/var/heimdal/m-key"
```

KDC の構築

- ３：管理用プリンシパルを追加

  kadmin -l add hrs/admin@ALLBSD.ORG

  - 典型的には UID/admin や UID/root という名前にする

  - 「プリンシパル」とは、鍵を持つユーザ名やホスト名を指す用語

- 鍵とプリンシパルは1:1関係

```
# kadmin -l add hrs/admin@ALLBSD.ORG
Max ticket life [1 day]:
Max renewable life [1 week]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
hrs/admin@ALLBSD.ORG's Password:
Verifying - hrs/admin@ALLBSD.ORG's Password:
```

KDC の構築

- ４：kdc と kadmind と kpasswdd を起動
  - rc.conf に書く
  - 分かりにくいので、近々変えます…

```
kerberos5_server_enable="YES"
kadmind5_server_enable="YES"
kpasswdd_server_enable="YES"
```

```
# service kerberos start
# service kadmind start
# service kpasswdd start
```

KDC の構築

- ４：kdc と kadmind と kpasswdd を起動
  - rc.conf に書く
  - 分かりにくいので、近々変えます…
  - （↓）こうなる予定

```
kdc_enable=“YES"
kadmind_enable="YES"
kpasswdd_enable="YES"
```

KDC の構築

- ５：kadmind の ACL を設定
  - kadmind は、KDC と同じマシンで動かす
  - 認証データベースの操作を行うためのデーモン
  - 認証にはもちろんKerberosを使う
- 認可（誰がアクセスできるのか）を決めておかないといけない

KDC の構築

- ５：kadmind の ACL を設定
  - kadmind は、KDC と同じマシンで動かす
  - 認証データベースの操作を行うためのデーモン
  - 認証にはもちろんKerberosを使う
  - 認可（誰がアクセスできるのか）を決めておかないといけない
  - /var/heimdal/kadmind.acl に書く
  - 書式：プリンシパル　アクション　対象プリンシパル

```
hrs/admin@ALLBSD.ORG all *
```

クライアントの設定

- クライアント設定に必要なもの
  - KDCの場所
  - デフォルトレルム
  - サービスプリンシパル名と鍵
- kadmind の例

kadmind

- KDCの場所
  - クライアントは KDC の場所を知らない
- デフォルトレルム設定
  - プリンシパル名を hrs とした時に、hrs@ALLBSD.ORG と解釈する
  - いつも @ 以降を書くなら、なくても動く
- どちらも /etc/krb5.conf か DNS に設定

/etc/krb5.conf

- すべてのクライアントに必要
- kadmind, kpasswdd のデーモンの場所も書ける

```
[libdefaults]
  default_realm = ALLBSD.ORG
[realms]
   ALLBSD.ORG = {
      kdc = 192.168.2.1:88
      admin_server = 192.168.2.1:749
      kpasswd_server = 192.168.2.1:464
   }
```

DNS

- SRV RR に指定
- 優先順位を指定して複数書ける
- kadmind は _kerberos-adm._tcp を使う
- デフォルトレルム名は、_kerberos に TXT で定義

```
_kerberos._tcp                  SRV     10 1 88 kdc.allbsd.org.
_kerberos._udp                  SRV     10 1 88 kdc.allbsd.org.
_kerberos._tcp                  SRV     20 1 88 kdc2.allbsd.org.
_kerberos._udp                  SRV     20 1 88 kdc2.allbsd.org.

_kpasswd._udp                   SRV     10 1 464 kdc.allbsd.org.
_kerberos-adm._tcp              SRV     10 1 749 kdc.allbsd.org.
_kerberos                       TXT     "ALLBSD.ORG"
```

どちらが良いか？

- DNS が管理できるなら、SRV RR を使うほうが圧倒的に楽
- /etc/krb5.conf は空でOK

kadmind

- kdc, kadmind, kpasswdd の起動

```
# service kerberos start
# service kadmind start
# service kpasswdd start
```

- /etc/rc.conf の編集を忘れずに

kadmind

- kadmin をリモートから使ってみる

```
% kadmin -p hrs/admin add hrs@ALLBSD.ORG
hrs/admin@ALLBSD.ORG's Password:
Max ticket life [1 day]:
Max renewable life [1 week]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
```

- hrs/admin@ALLBSD.ORG で認証して、KDCデータベースを編集

kadminユーティリティ

- プリンシパルの追加

```
% kadmin -p hrs/admin add hrs@ALLBSD.ORG
```

- プリンシパルの削除

```
% kadmin -p hrs/admin delete hrs@ALLBSD.ORG
```

- プリンシパルの表示

```
% kadmin -p hrs/admin get hrs@ALLBSD.ORG
```

- プリンシパル属性情報の変更

```
% kadmin -p hrs/admin modify hrs@ALLBSD.ORG
```

- 鍵の取り出し

```
% kadmin -p hrs/admin ext_keytab -k /tmp/keytab hrs@ALLBSD.ORG
```

kpasswdユーティリティ

- passwd(1) と同じように、プリンシパルのパスワードを変更できる

```
% kpasswd hrs/admin@ALLBSD.ORG
```

注意

- Kerberos はリプレイ攻撃を防ぐため、タイムスタンプを使う
- 時刻が合ってないと動きません
- NTPをちゃんと設定しましょう
  - DDoS の標的にならないように！

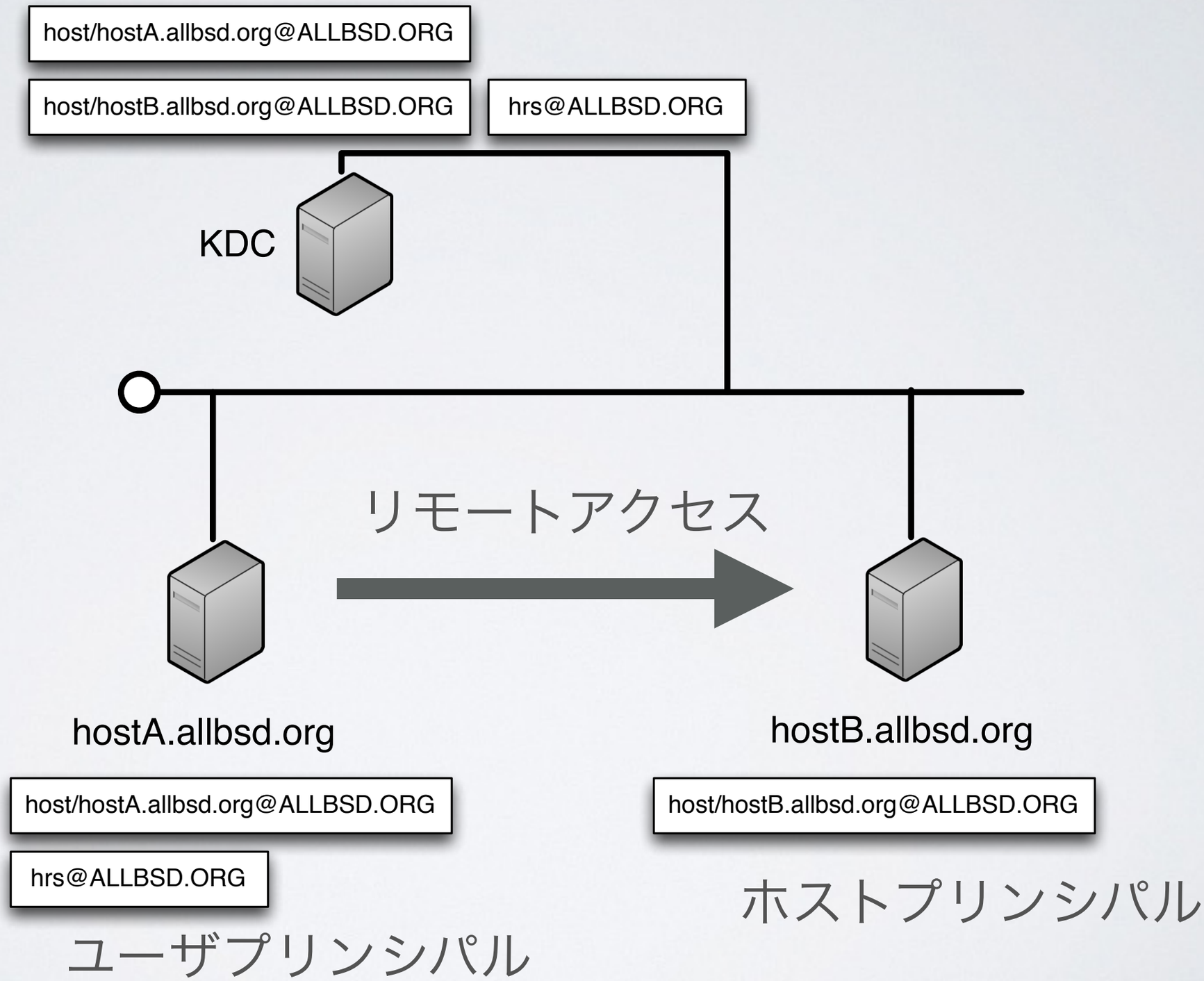
5分休憩

・次は個々のクライアントアプリケーションへの応用です。

OpenSSH

- リモートログインに使ってみよう
- 復習
  - Kerberos は 2 者間の認証を行う
  - 認証をする主体には必ず鍵があり、KDCは全部の鍵を持っている
  - リモートログインの主体は「ユーザ」と「ホスト」

OpenSSH

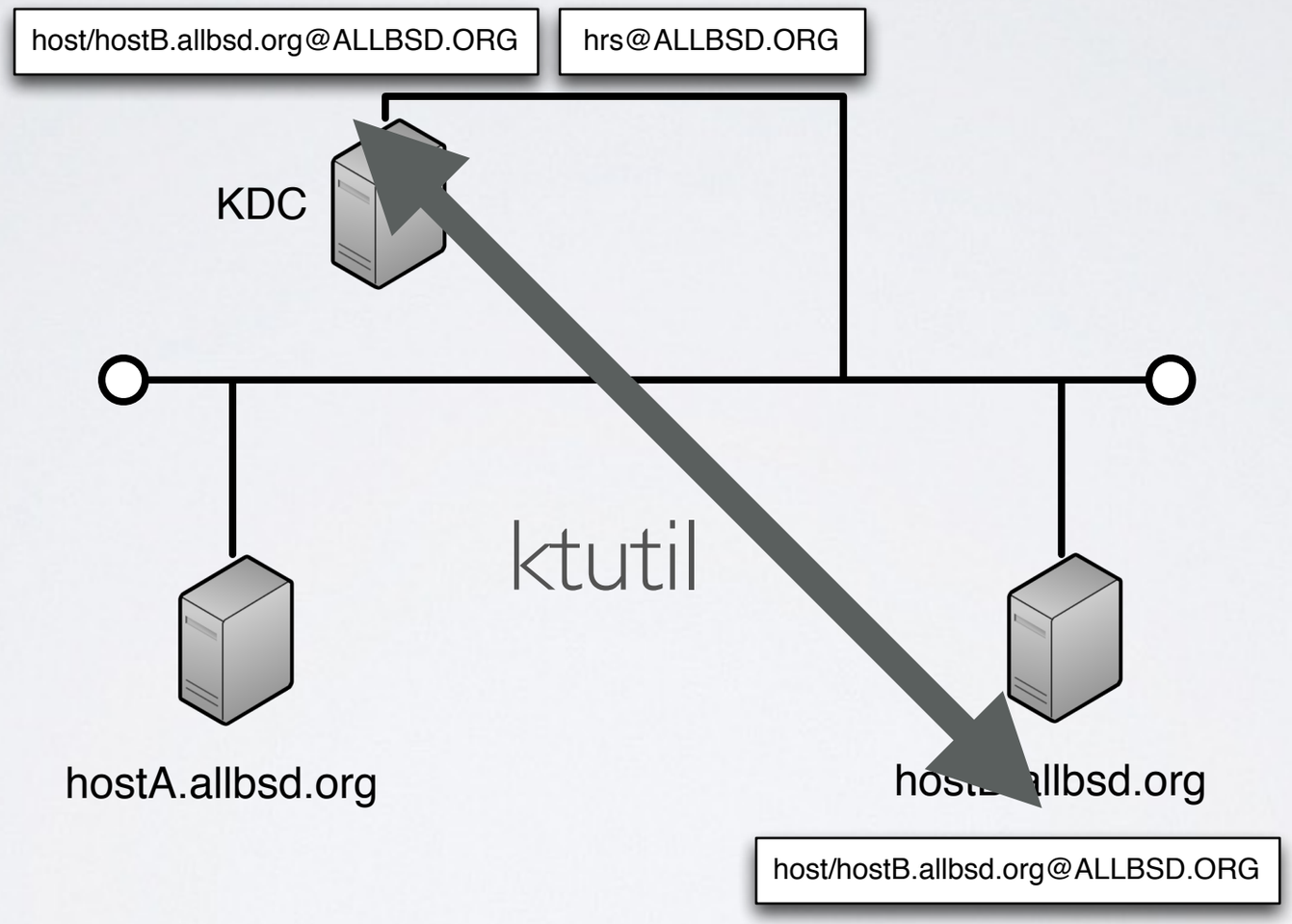

host/hostA.allbsd.org@ALLBSD.ORG
host/hostB.allbsd.org@ALLBSD.ORG
hrs@ALLBSD.ORG
KDC
リモートアクセス
hostA.allbsd.org
hostB.allbsd.org
host/hostA.allbsd.org@ALLBSD.ORG
hrs@ALLBSD.ORG
host/hostB.allbsd.org@ALLBSD.ORG
ユーザプリンシパル
ホストプリンシパル

OpenSSH

- ステップ
  - hostBにホスト鍵をつくってKDCに登録。
  - OpenSSHに、Kerberosを使うように設定。
  - OpenSSHの認可データベースをチェック。

OpenSSH

- ホストプリンシパルの生成と登録



OpenSSH

- ホストプリンシパルの生成と登録
  - KDC と hostB の両方の置かないといけない
  - /etc/krb5.keytab を使う（root だけが読める）
  - コマンド一発

```
# ktutil get -p hrs/admin host/`hostname`@ALLBSD.ORG
```

- ポイント
  - ホストプリンシパルには「host/FQDN@realm」という名前を付ける。
  - /etc/krb5.keytab はrootで書き込むので、root権限で実行すること。

OpenSSH

- ktutil ユーティリティ
  - ローカルの鍵ファイルを管理するためのツール
  - ちゃんとできてるか確認

```
# ktutil list
FILE:/etc/krb5.keytab:

Vno  Type                      Principal                                Aliases
  1  des-cbc-md5                host/hostB.allbsd.org@ALLBSD.ORG
  1  des-cbc-md4                host/hostB.allbsd.org@ALLBSD.ORG
  1  des-cbc-crc                host/hostB.allbsd.org@ALLBSD.ORG
  1  aes256-cts-hmac-sha1-96    host/hostB.allbsd.org@ALLBSD.ORG
  1  des3-cbc-sha1              host/hostB.allbsd.org@ALLBSD.ORG
  1  arcfour-hmac-md5           host/hostB.allbsd.org@ALLBSD.ORG
```

OpenSSH

- OpenSSH の設定
  - GSSAPIAuthentication=yes を追加（サーバ・クライアントの両方）
    - /etc/ssh/sshd_config
    - /etc/ssh/ssh_config
    - $HOME/.ssh/config
    - /etc/rc.conf
- 他に設定の必要なし

OpenSSH

- 使ってみよう
  - まずはユーザプリンシパルの指定

```
% kinit
hrs@ALLBSD.ORG's Password:

% klist
Credentials cache: FILE:/tmp/krb5cc_20001
        Principal: hrs@ALLBSD.ORG

  Issued                Expires               Principal
Mar 13 05:24:40 2014  Mar 13 15:24:40 2014  krbtgt/ALLBSD.ORG@ALLBSD.ORG
```

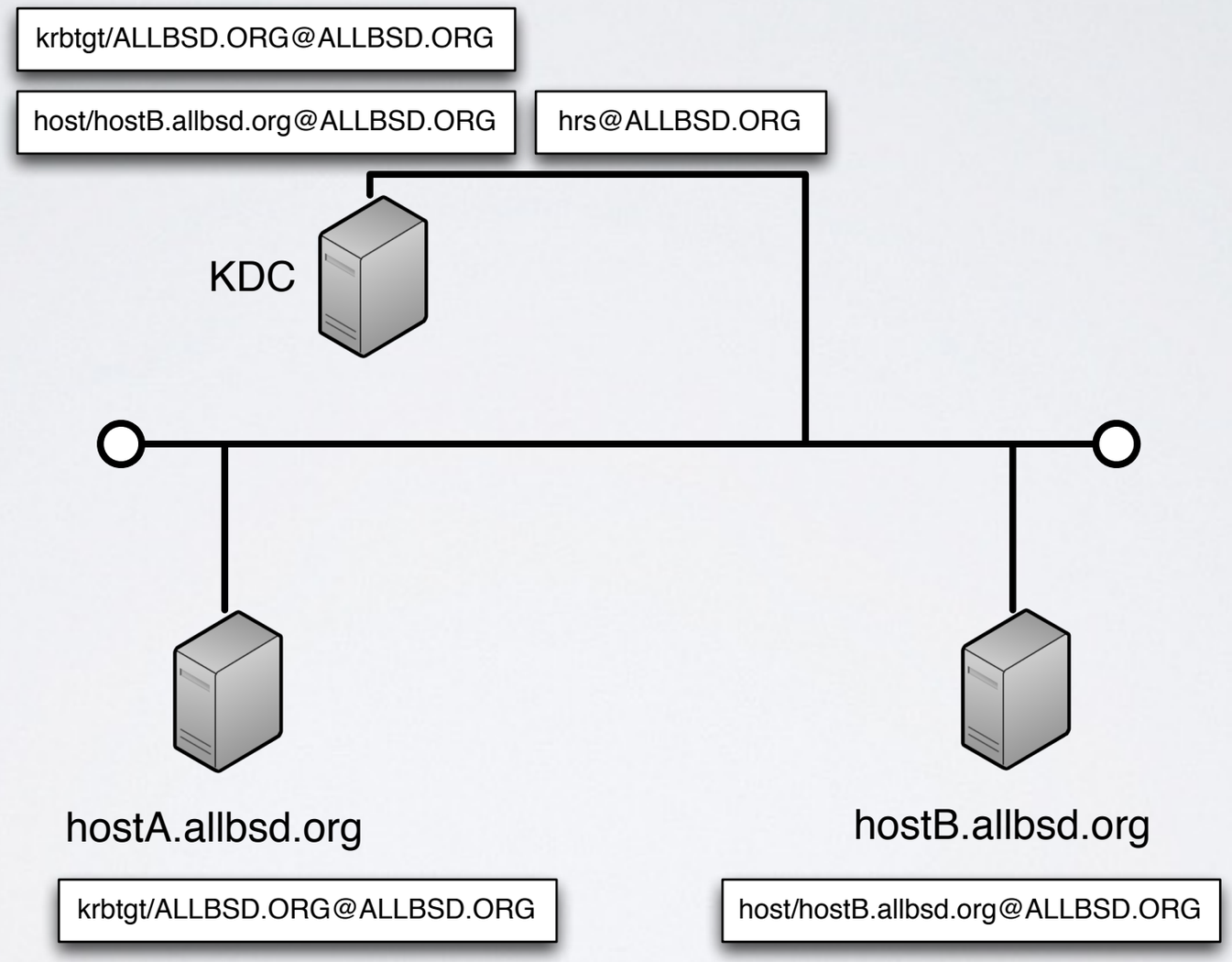
OpenSSH

- 使ってみよう
  - kinit: ユーザ鍵を使ってTGTを取ってくる操作。
    一回行うと、以降の認証はパスワードの入力が不要に。
  - klist: 自分の持っているチケットを表示するコマンド

```
% kinit
hrs@ALLBSD.ORG's Password:

% klist
Credentials cache: FILE:/tmp/krb5cc_20001
        Principal: hrs@ALLBSD.ORG

  Issued                Expires               Principal
Mar 13 05:24:40 2014  Mar 13 15:24:40 2014  krbtgt/ALLBSD.ORG@ALLBSD.ORG
```

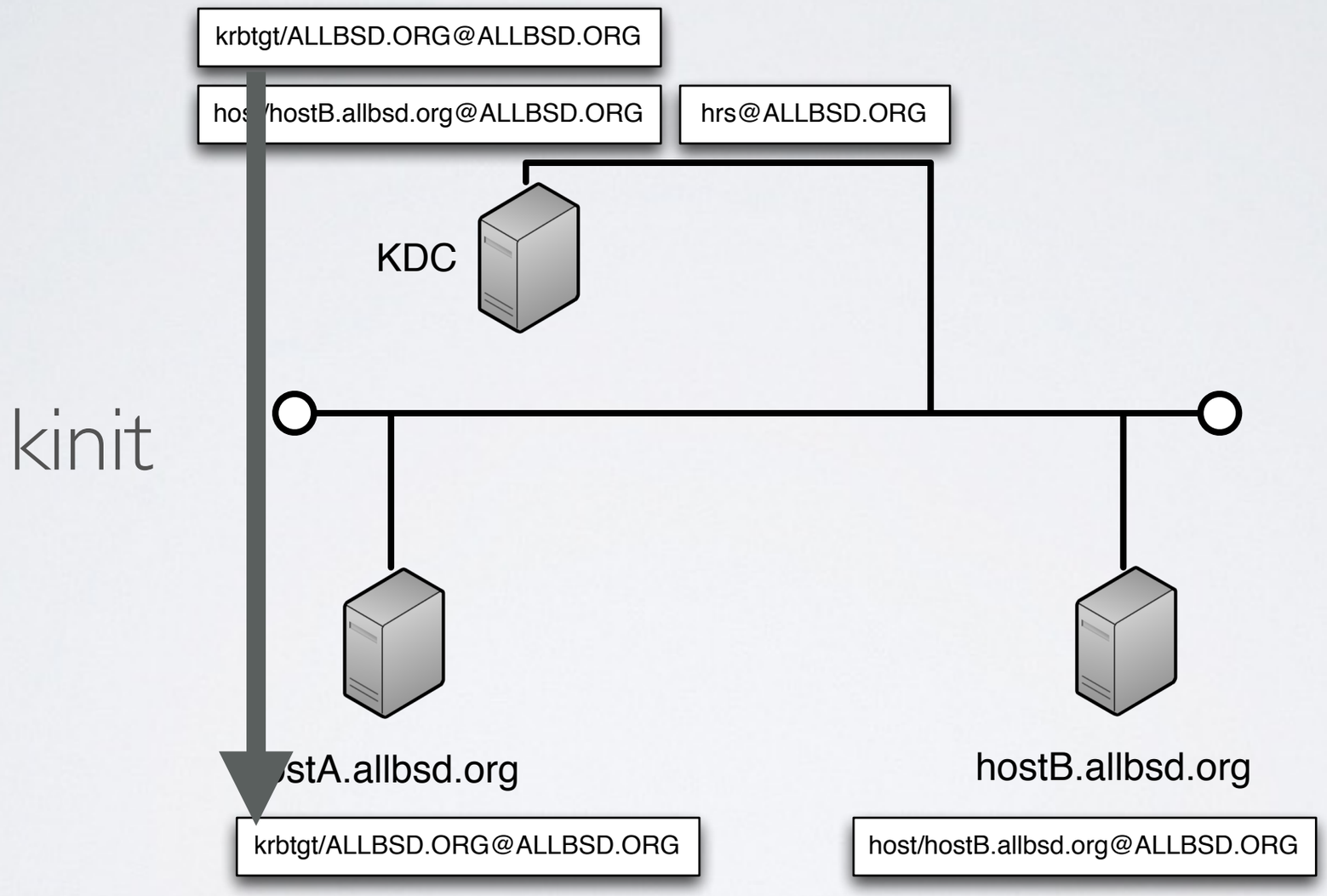
OpenSSH

・もう一回復習



OpenSSH

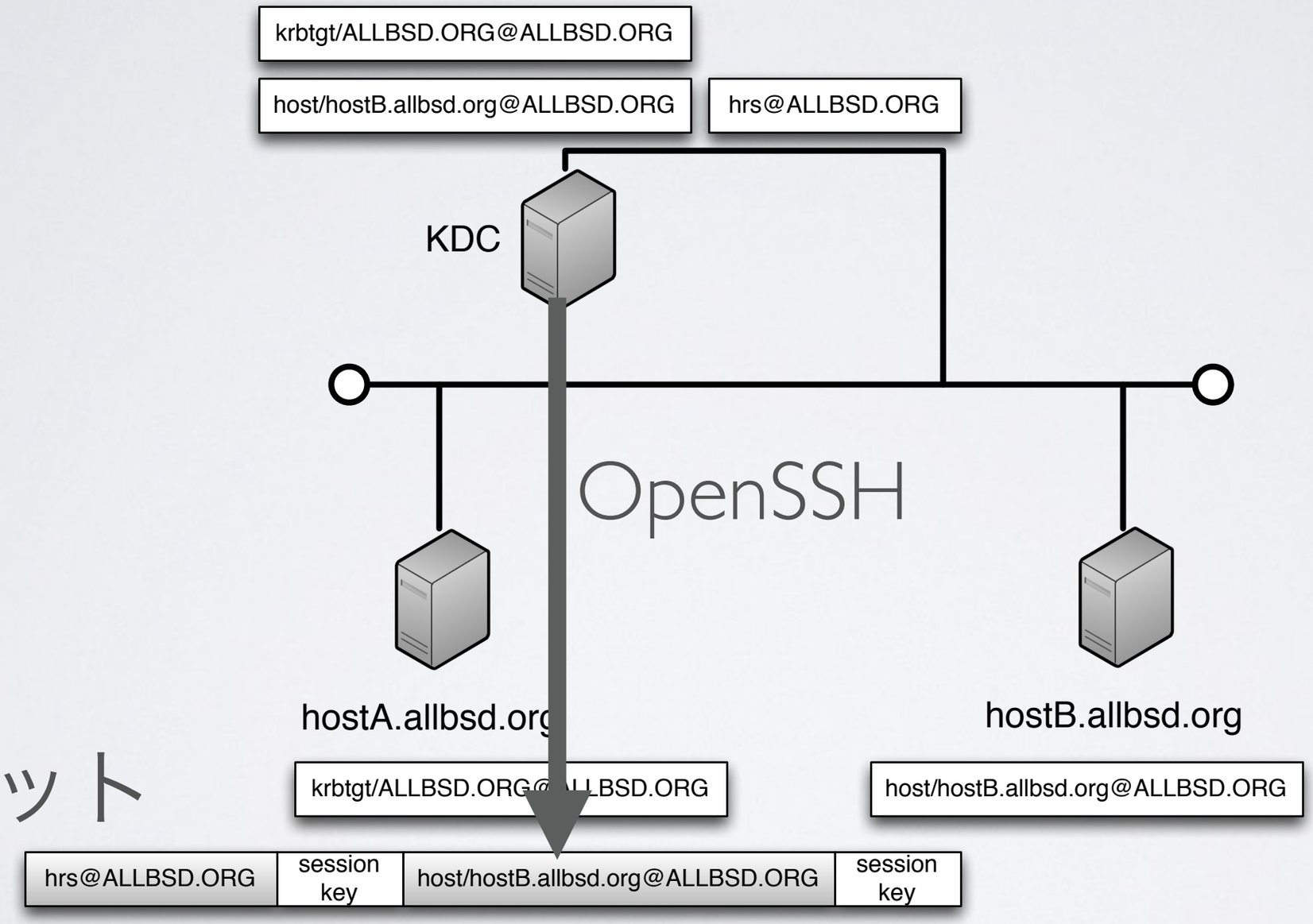
・もう一回復習



TGTはKDCから他のチケットを取るのに必要

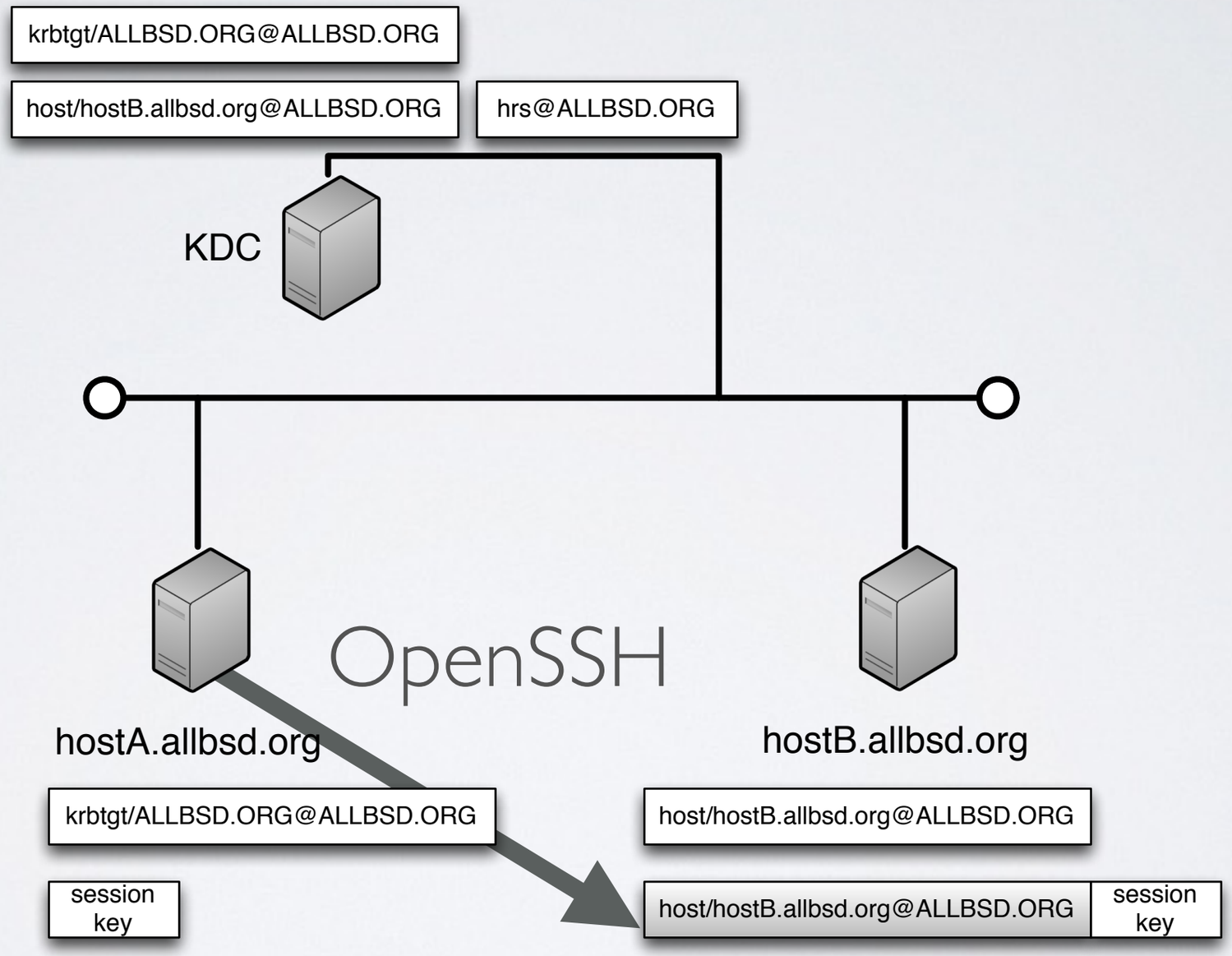
OpenSSH

・もう一回復習



OpenSSH

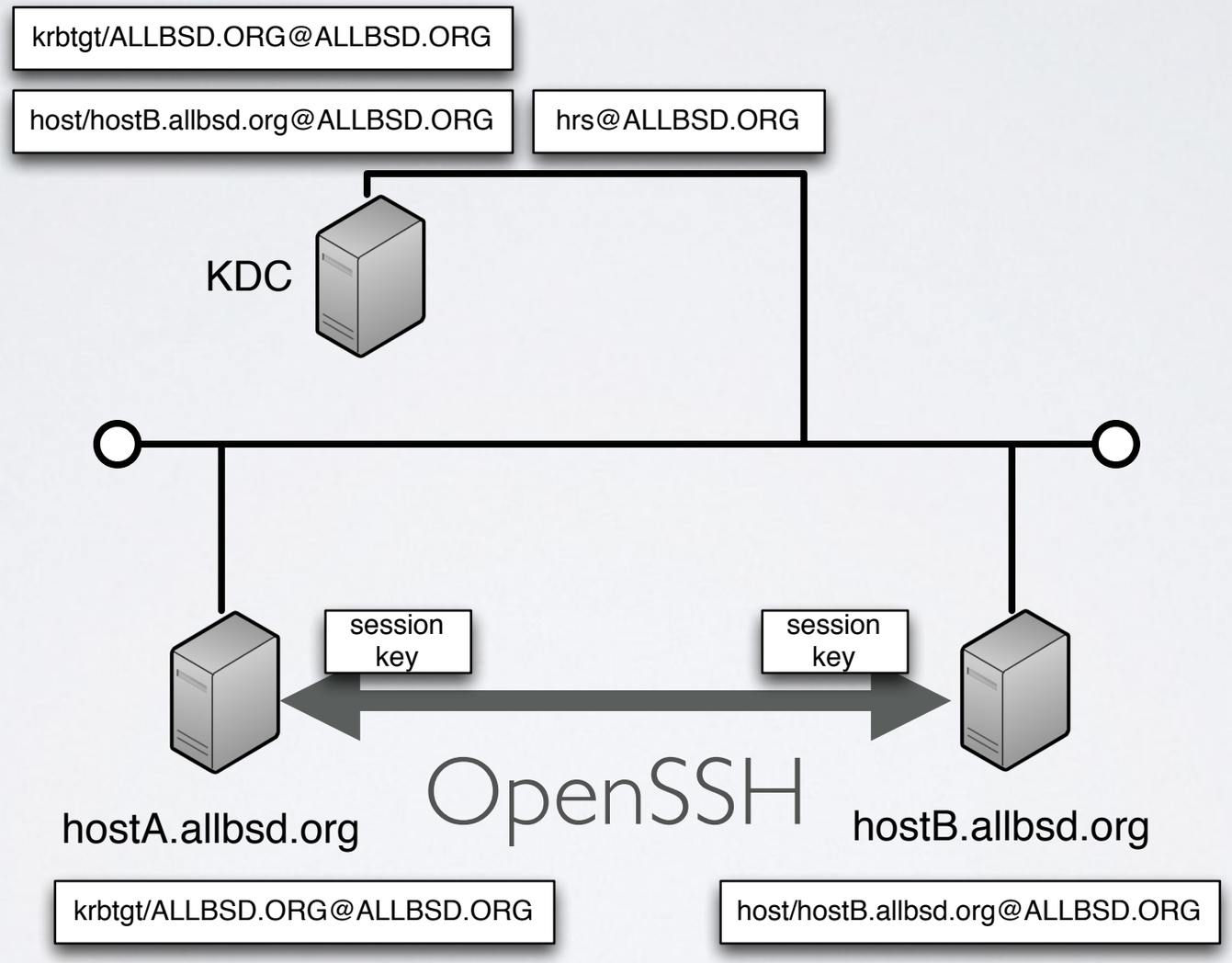
・もう一回復習



セッション鍵を送る

OpenSSH

・もう一回復習



OpenSSH

- 使ってみよう
- あとはいつもどおり ssh でアクセス

```
% ssh -4v hostB.allbsd.org
OpenSSH_6.3, OpenSSL 1.0.1e-freebsd 11 Feb 2013
debug1: Reading configuration data /home/hrs/.ssh/config
. . .
debug1: Connecting to hostB.allbsd.org [192.168.2.2] port 22.
debug1: Connection established.
. . .
debug1: Authentications that can continue: publickey,gssapi-with-mic
debug1: Next authentication method: gssapi-with-mic
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Authentication succeeded (gssapi-with-mic).
Authenticated to hostB.allbsd.org ([192.168.2.2]:22).
```

覚えておくべきこと

- 認証が必要なのは kinit でTGTを取得する時だけ
- TGTは /tmp/krb5cc_${UID} に保存される。
  場所を変えることも可能(KRB5CCNAME)
- TGTには有効期限があり、切れると要パスワード再入力

OpenSSH

- 認可はどうやってやるの？
  - 指定しないと、Kerberos 認証できたユーザは全員ログインできる
  - AllowUsers, AllowGroups オプションを使いましょう
  - プリンシパル名とUIDとの対応は、~/.k5login というファイルを使う

```
% cat ~/.k5login
hrs@ALLBSD.ORG
foo@ALLBSD.ORG
```

- foo@ALLBSD.ORG のチケットを持っているユーザは、
  ssh hrs@hostB.allbsd.org でログインできる
- root ユーザでログインしたい場合などに使うことが多い

OpenSSH

- ssh-agent のようなことはできる？
  - ログイン先にチケットを転送する機能がある
  - ssh に GSSAPIDlegateCredentials=yes を設定（デフォルトはno）

OpenSSH

- 公開鍵認証と何が違うのか
  - ホストプリンシパルとユーザプリンシパルによる認証
    - ちゃんと登録されていないと認証されない
    - パスワードの revoke が確実にできる
- 置き換えるものではなく、適材適所

login

- ローカルログインでも使ってみよう
- PAMを使って、認証先を切り替える

```
% cat /etc/pam.d/system
# auth
auth            sufficient      pam_opie.so             no_warn no_fake_prompts
auth            requisite       pam_opieaccess.so       no_warn allow_local
auth            sufficient      pam_krb5.so             no_warn try_first_pass
#auth           sufficient      pam_ssh.so              no_warn try_first_pass
auth            required        pam_unix.so             no_warn try_first_pass
nullok
```

- 注意：ホストプリンシパルがないと認証が成功しません！

login

- ログインしただけで、自動的にTGTを取ってくれる

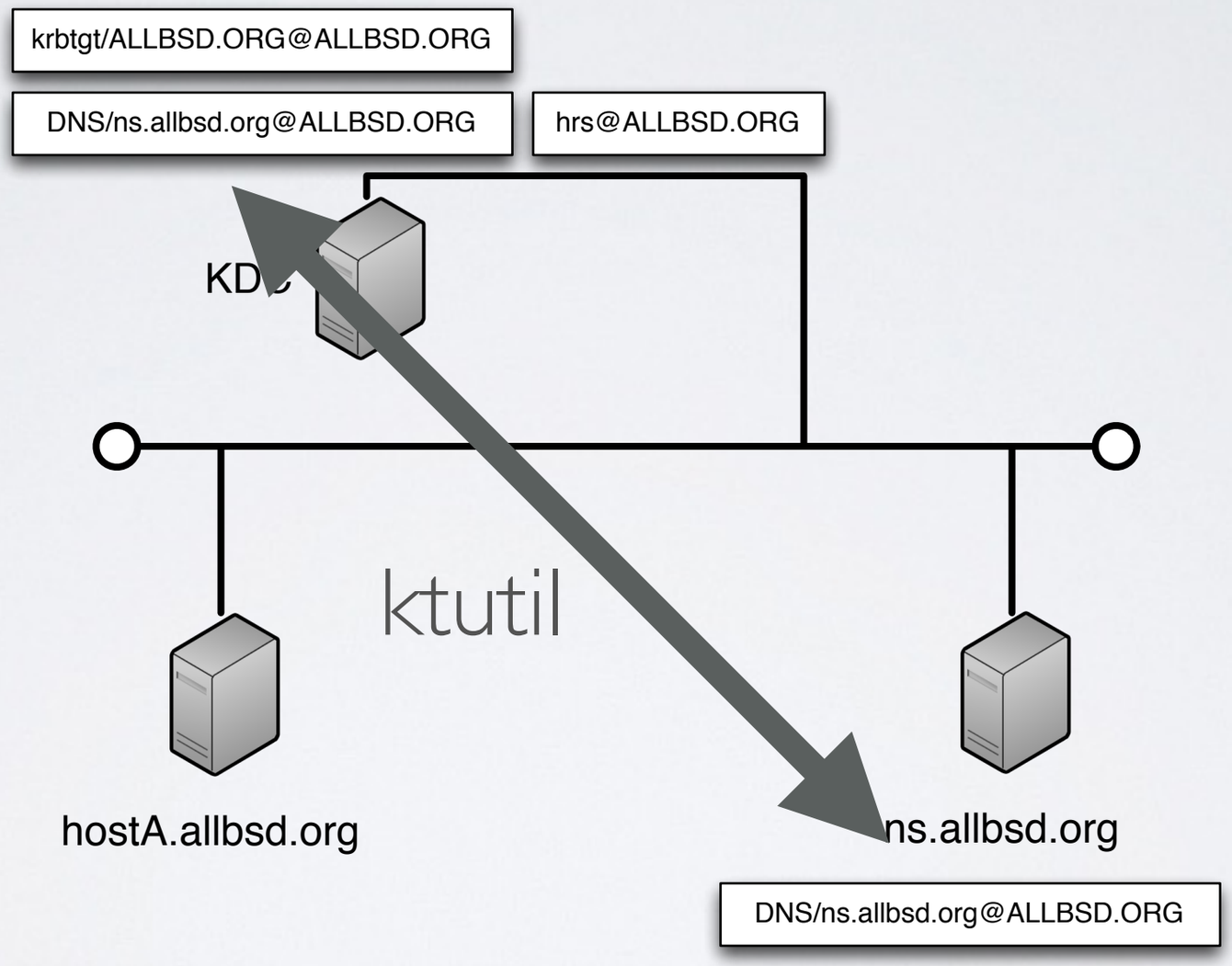
```
login: hrs
Password:
. . .

% klist
Credentials cache: FILE:/tmp/krb5cc_20001
        Principal: hrs@ALLBSD.ORG

  Issued                  Expires                 Principal
Mar 13 06:08:39 2014  Mar 13 16:08:39 2014  host/hostB.allbsd.org@ALLBSD.ORG
Mar 13 06:08:39 2014  Mar 13 16:08:39 2014  krbtgt/ALLBSD.ORG@ALLBSD.ORG
```

BIND

- TSIG認証をkerberosで行うことが可能
- ここで問題
  - 今までの例は、kinit を手動で使っていた
  - TSIG認証はサーバ間で使うこともあるが、どうやって設定するの？
- nsupdate を使った例

BIND

・もう一度復習


krbtgt/ALLBSD.ORG@ALLBSD.ORG
DNS/ns.allbsd.org@ALLBSD.ORG
hrs@ALLBSD.ORG
KDC
ktutil
hostA.allbsd.org
ns.allbsd.org
DNS/ns.allbsd.org@ALLBSD.ORG

BIND

- BIND は、サービスプリンシパルとして host/FQDN ではなく「DNS/FQDN」を使う
- このプリンシパルは、named しか使わないので、/etc/krb5.keytab ではなく、別のファイル /etc/namedb/named.keytab に保存する

```
# ktutil -k /etc/namedb/named.keytab get —p hrs/admin DNS/`hostname`
# chown bind /etc/namedb/named.keytab
```

BIND

- BIND の DDNS に GSS-TSIG を設定する
  - options セクションに tkey-gssapi-keytab
  - zone セクションに grant 文を入れる

```
options {
  tkey-gssapi-keytab "/etc/namedb/named.keytab";
};

zone “allbsd.org” {
. . .
                update-policy {
                        grant hrs@ALLBSD.ORG wildcard * ANY;
                        grant "DNS/ns.allbsd.org@ALLBSD.ORG" wildcard * ANY;
                };
}
```

BIND

- **クライアント**
  - kinit して TGT を取得
  - nsupdate -g を実行（もしくは gsstsig コマンドを実行）

```
gsstsig
realm ALLBSD.ORG
server 133.31.130.32
local 133.31.130.32
zone hrslab.org.
update add www.hrslab.org. 600 A 133.31.130.32
show
send
```

BIND

- nsupdate を自動でやりたい。たとえば cron で kinit するには？
  - 鍵をあらかじめKDCから取り出しておけば良い
  - 注意：サービス単位で鍵をつくり、鍵を守ること。

```
% kadmin ext_keytab -k /tmp/foo.keytab foo@ALLBSD.ORG
% kinit -t /tmp/foo.keytab （パスワードは不要）
```

- ↑の kinit を crontab に書けば無人で動く

ここまでで学んだこと

- **Kerberos を使うには、**
  使う側・サービス側の両方のプリンシパルが必要
  - SSH はユーザとホストプリンシパル(host/FQDN)
  - BIND はユーザとサービスプリンシパル(DNS/FQDN)
    - 「どのプリンシパルを使うのか」はマニュアルに書いてある
- **サービス側には認可データベースがある**
  - 認証された後、サービスを許可するかどうか
  - SSHは何も設定しなければ/etc/passwdにあるかどうかだけを見る
- **時刻同期が重要**

ここまでで学んだこと

- **プリンシパルの鍵は、KDCから取り出すことができる**
  - 取り出すと、パスワードは要らなくなる
    - kinit するのと、鍵を取り出すのとは違う
  - 実はサービスプリンシパルの大半は取り出して使っている
    - SSH サーバは/etc/krb5.keytab を読んでいる
    - BINDの例では /etc/namedb/named.keytab を使った
- パスワードレスで使いたい時には取り出す
- 盗まれると終わるので、取り扱いは十分注意

Apache, curl

- **HTTPの認証をKerberosで**
  - HTTP Basic 認証を KDC のパスワードで行う
  - SPNEGO: 最もまともな Kerberos 認証 over HTTP

Apache, curl

- **Apache の設定**
  - mod_auth_kreb を追加し、AuthTypeを Kerberos にする
  - サービスプリンシパルとして HTTP/FQDNをつくる必要あり
  - この例では /usr/local/etc/httpd.keytab に置いている

```
LoadModule auth_kerb_module   libexec/apache22/mod_auth_kerb.so

AuthType Kerberos
Authname "Kerberos Login"
KrbAuthRealms ALLBSD.ORG
KrbServiceName HTTP/www.allbsd.org
Krb5Keytab /usr/local/etc/httpd.keytab
KrbMethodNegotiate on
KrbMethodK5Passwd off
```

Apache, curl

- curl でアクセスしてみる

```
% curl –negotiate http://hrs@www.allbsd.org
```

- ユーザ名の指定が必要

Apache, curl

- Apache の設定
  - KrbMethodNegotiate を off にし、KrbMethodK5Passwd を on にすると、単なるパスワード認証になる

```
LoadModule auth_kerb_module   libexec/apache22/mod_auth_kerb.so

AuthType Kerberos
Authname "Kerberos Login"
KrbAuthRealms ALLBSD.ORG
KrbServiceName HTTP/www.allbsd.org
Krb5Keytab /usr/local/etc/httpd.keytab
KrbMethodNegotiate on
KrbMethodK5Passwd off
```

KDC 運用

- KDC のログはどうやって管理すれば？
- KDC が落ちると認証できない
- パケットフィルタの設定は？

ログ

- デフォルトのログ
  - /var/heimdal/kdc.log
  - syslog (INFO)
  - あまり使いやすくないので、全部 syslog に飛ばすのが良い

/etc/krb5.conf

```
[logging]
        kdc = SYSLOG
        kadmind = SYSLOG
        kpasswdd = SYSLOG
        default = SYSLOG
```

/etc/syslog.conf

```
!kdc,kadmind,kpasswdd
*.*                                             /var/log/kdc.log
```

スレーブサーバ

- KDC が落ちると認証できない → スレーブサーバをつくる
- 方法は2つ: hprop （全転送）と iprop（差分転送）
  - iprop のほうが新しい
- ステップ
  - KDCを同じマスタ暗号鍵でつくる
  - データベースを同期させる

iprop 同期の設定

- **マスタ鍵が同じKDCをつくる**
  - kadmin -l init した後に m-key をマスタから手動でコピー
- **サービスプリンシパルをつくる**
  - iprop/<マスタホスト名>
  - iprop/<スレーブホスト名>
- **認可データベースをつくる（サーバのみ）**
  - /var/heimdal/slaves
  - スレーブのプリンシパル名を行単位で書く

iprop 同期の設定

- **ipropd-master を起動（サーバ）**
- **ipropd-slave を起動（クライアント）**
  - 2121/tcp で通信して同期

```
# /usr/libexec/ipropd-master –detach
```

```
# /usr/libexec/ipropd-slave –detach <マスタサーバのホスト名>
```

- /usr/libexec/ipropd-master と /usr/libexec/ipropd-slave
  （rc.d/ipropd_master, rc.d/ipropd_slave を追加する予定）

iprop 同期の設定

- クライアントの接続状況は、/var/heimdal/slaves-stats に記録される
- 接続が切れても根気よく再接続を試みる

```
# cat /var/heimdal/slaves-stats
Status for slaves, last updated: 2014-03-13T11:18:48

Master version: 157

Name                                        Address                 Version  Status  Last Seen
iprop/hrshome.allbsd.org@ALLBSD.ORG  IPv4:192.168.2.4        157  Up      2014-03-13T11:18:18
```

- 運用上の注意
  - YAT message のログ（サーバ側）は無視して良し

スレーブサーバ

- **KDC が 2 つになったら**
  - SRV RR を追加する
  - /etc/krb5.conf に kdc= 行を追加する
- あとは自動的にフェイルオーバする

通信の詳細

- **使うポート番号まとめ**
  - kdc: 88/tcp, 88/udp
  - kpasswdd: 464/tcp
  - kadmind: 749/tcp
  - ipropd-master: 2121/tcp
  - NATを越えることは可能
- **HTTP プロキシを使うことも可能**
  - /etc/krb5.conf に指定（相互運用性は高くない）

```
[libdefaults]
  http_proxy = http://proxy.allbsd.org/
```

簡単な運用方法

- KDC 側
  - kdc, kadmind, kpasswdd を起動する
  - SRV RR に、レルム名とサービスIPアドレスを指定する
- クライアント側
  - SRV RR があれば、kinit するだけで良い
  - PAM を使って kinit させるのが簡単

```
    auth            sufficient      pam_krb5.so                 no_warn try_first_pass
    account         required        pam_krb5.so
```

簡単な運用方法

- **SSH の公開鍵認証と組み合わせて適材適所**
  - 新しいマシンを追加しても、パスワードハッシュを追加する必要性を極力抑えることができる。
  - 鍵の revoke が確実にできる。
  - 数百台のマシンに authorized_keys ファイルをコピーする、なんてのは悪夢です。
  - ホスト鍵が登録されている相手しか認証は成功しないので、KDC上の鍵を潰せば、すべての鍵は無効に

さらなる応用は？

- **Kerberos のバックエンドデータベースをLDAP化**
  - LDAP認証と統合できる
  - ActiveDirectory は KDC + LDAP
    - UNIX から KDC として使うことも可能 （Sambaなど）
- **他のアプリケーション**
  - アプリケーションが GSSAPI に対応しているかどうか調べよう
  - 対応していれば、Kerberos で認証できる可能性が高い
  - SASL は GSSAPI を使うことができる

おしまい

・質問はありますか？