

暗号プリミティブに基づく機密データフロー解析

○林 良太郎 中西 福友 橋本 幹生
(株式会社 東芝 研究開発センター)



東芝グループは、持続可能な
地球の未来に貢献します。

もくじ

- ソフトウェア保護技術
- DFITS (Data Flow Isolation Technology for Security)
- 機密性検証
- まとめ

もくじ

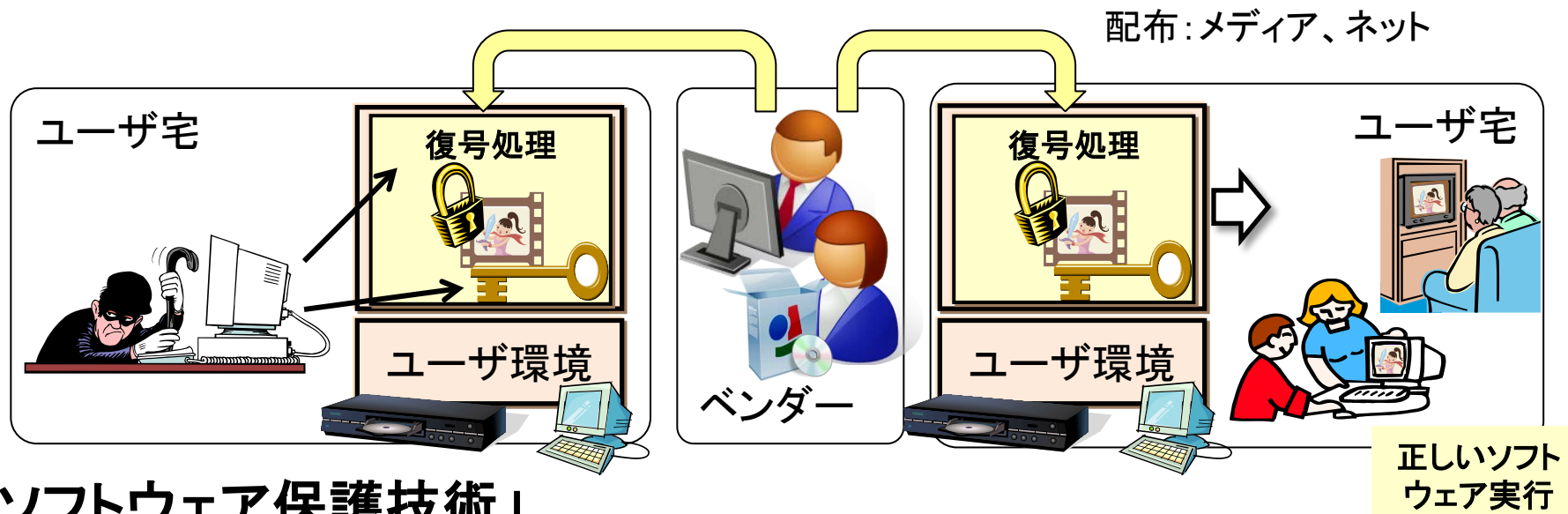
- **ソフトウェア保護技術**
- **DFITS (Data Flow Isolation Technology for Security)**
- **機密性検証**
- **まとめ**

ソフトウェア保護技術とは

- ・ ユーザ環境で実行するソフトウェアの内部に、ユーザに秘密とすべき情報が含まれる

例) ブルーレイディスクプレイヤーの場合

- データ(鍵情報, 保護コンテンツ)
- 自社独自アルゴリズム・ノウハウ



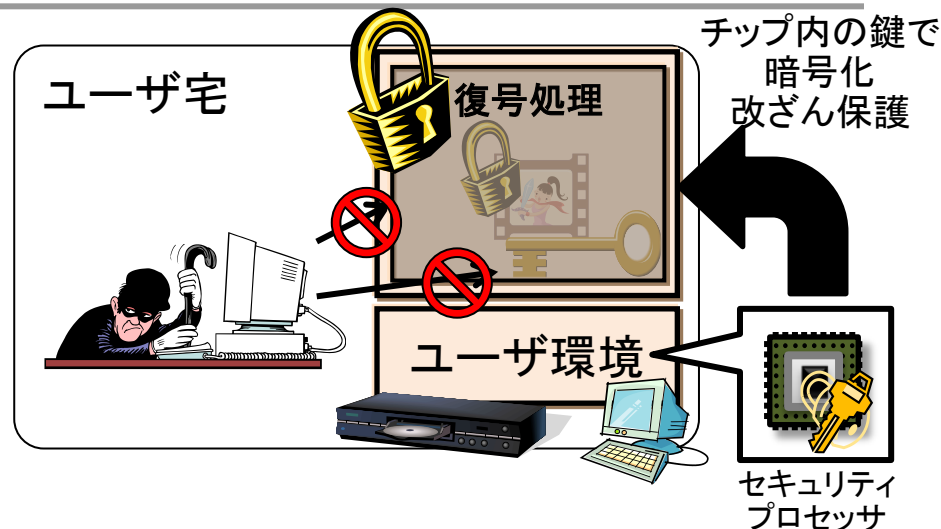
「ソフトウェア保護技術」

- これらの情報を「悪意のある」ユーザから秘匿するための技術
- ただ単に秘匿にするだけでなく、正常実行可能にすることが必須

ソフトウェア保護手段の例

• 専用ハードウェア

- セキュリティプロセッサ (XOM, AEGIS, LMSP™ など)
 - プロセッサチップ内の鍵による (メモリ上) データの暗号化
 - チップ内で復号・実行
 - 悪意のあるユーザは、保護されたデータを読み取ったり改ざんしたりできない



• ソフトウェア難読化

- ソフトウェアを解析 (リバースエンジニアリング) しづらくすることで、以下を把握することを困難にする
 - どこで機密情報を処理しているのか？
 - どのようなアルゴリズムを使っているのか？
- 部分的に難読化することも可能

ソフトウェア保護実現に必要なこと:How と What

- **How: どうやって保護するか？**
 - ソフトウェア難読化, 専用ハードウェアなどの手段あり
- **What: 何を保護するか？(どれが機密情報なのか？)**
 - 仕様書に記載されている Confidential Information だけでは不十分
 - 一時変数もすべて保護しなければならない
 - 実装まで踏み込まなければならない
 - ソフトウェアが持つ情報を何もかも守ればよい？ → No
 - ソフトウェアの情報が一切漏れない＝入出力の無いソフトウェア
 - 外部との入出力は必須
 - 過度に保護しすぎると実行速度が遅くなる
 - ソフトウェア保護技術(再掲): 機密情報を「悪意のある」ユーザから秘匿する.
ただし, ただ単に秘匿にするだけでなく、正常実行可能にすることが必須

⇒ **DFITS: Data Flow Isolation Technology for Security**

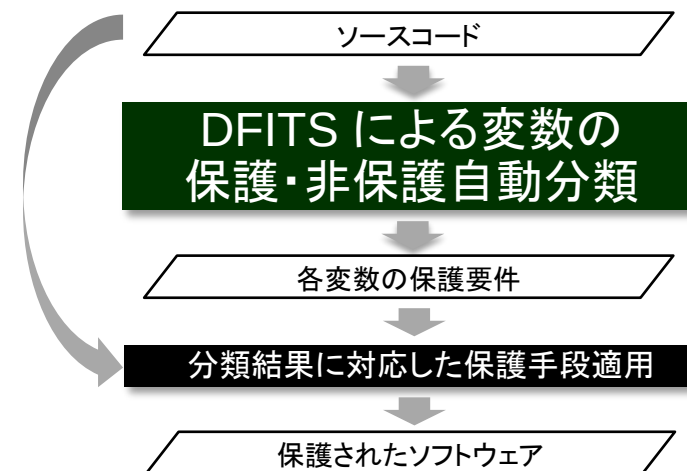
もくじ

- ソフトウェア保護技術
- **DFITS (Data Flow Isolation Technology for Security)**
- 機密性検証
- まとめ

DFITS™ (Data Flow Isolation Technology for Security)

• DFITS

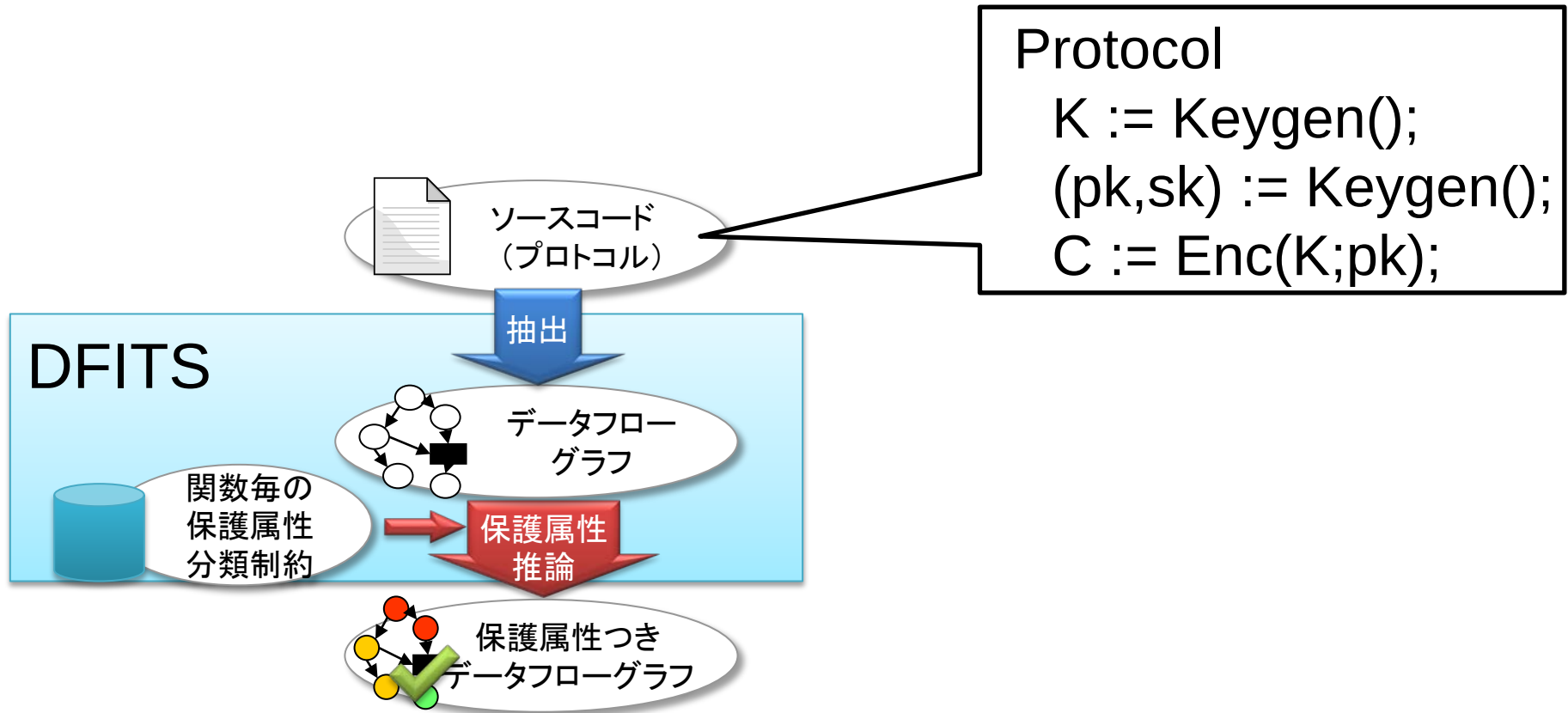
- 入力: プログラムソースコード
- 出力: プログラム中の各変数の保護要件
 - どれを守ればよいか分かる
 - 非安全なデータフローがあればエラーを出力
- 試作システム
 - C/C++ ソースコードを(条件付きで)解析可



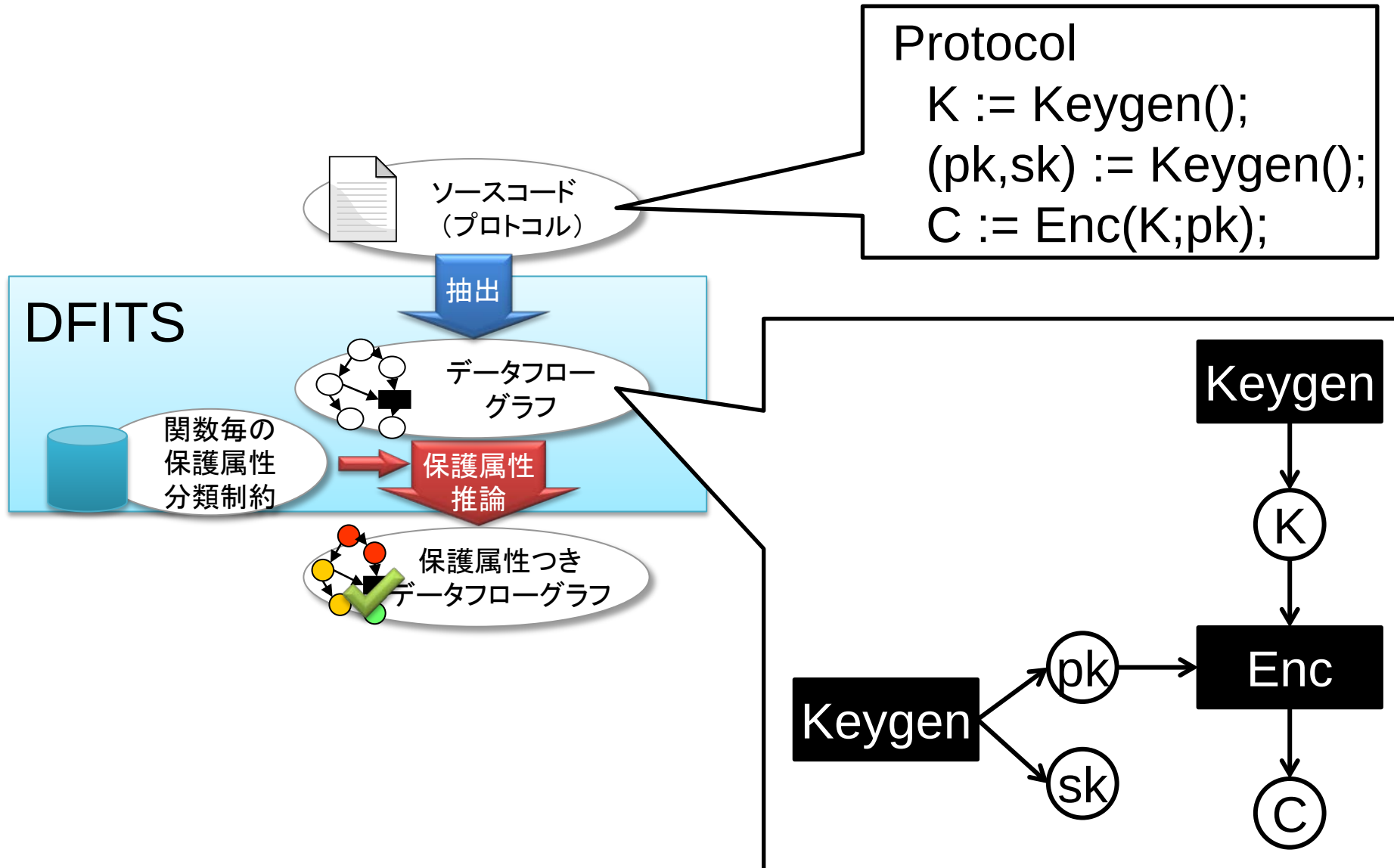
• 特徴

- 機密性の観点で分類(完全性は今後)
- プログラムにおける(機密)データの流れ(データフロー)に注目
- 保護・非保護判定の手がかり: プログラム中の暗号プリミティブ
 - 例) 秘密鍵は機密性必須, 公開鍵や暗号文は機密性保護不要...
- シンプルな手法

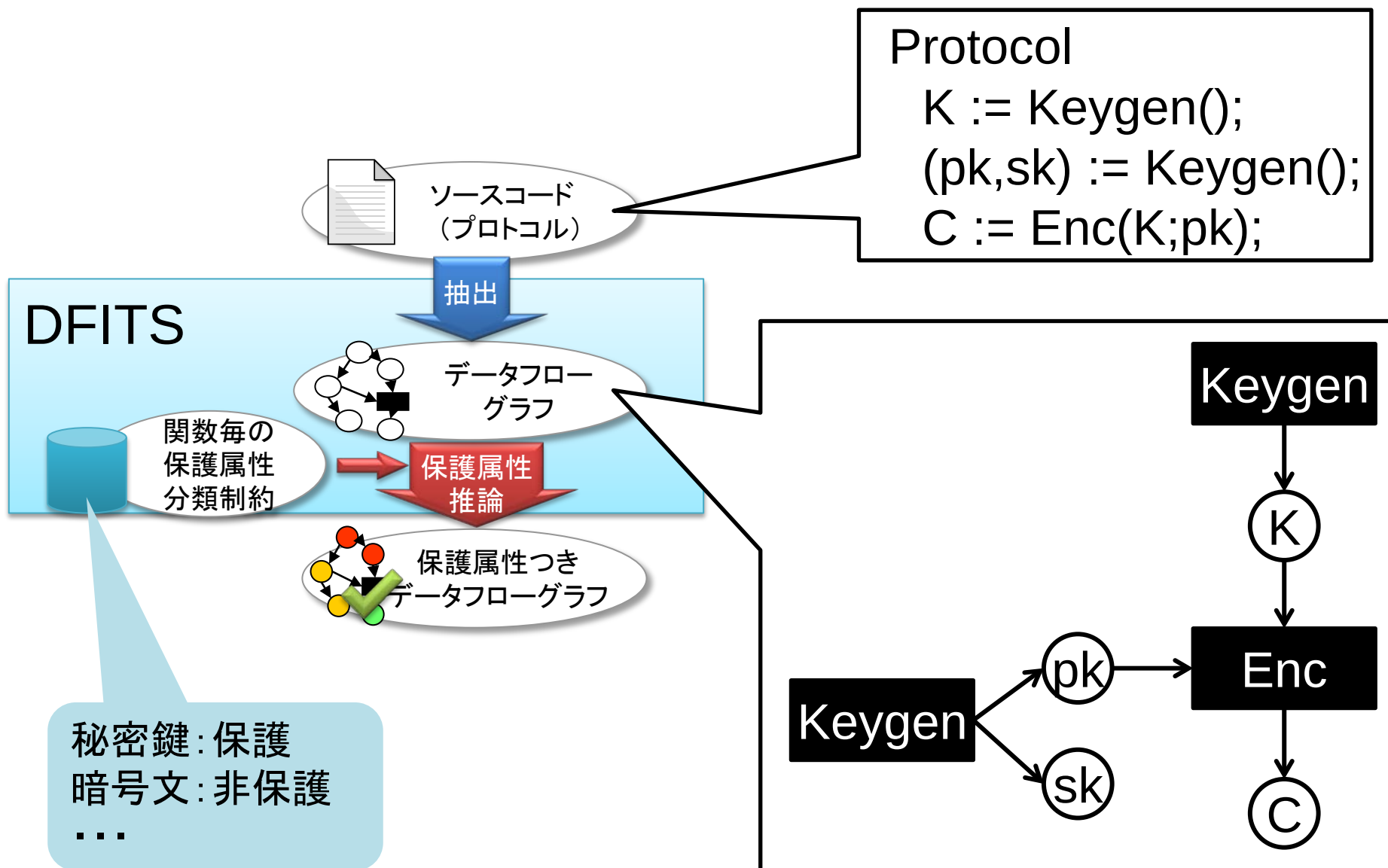
イメージ



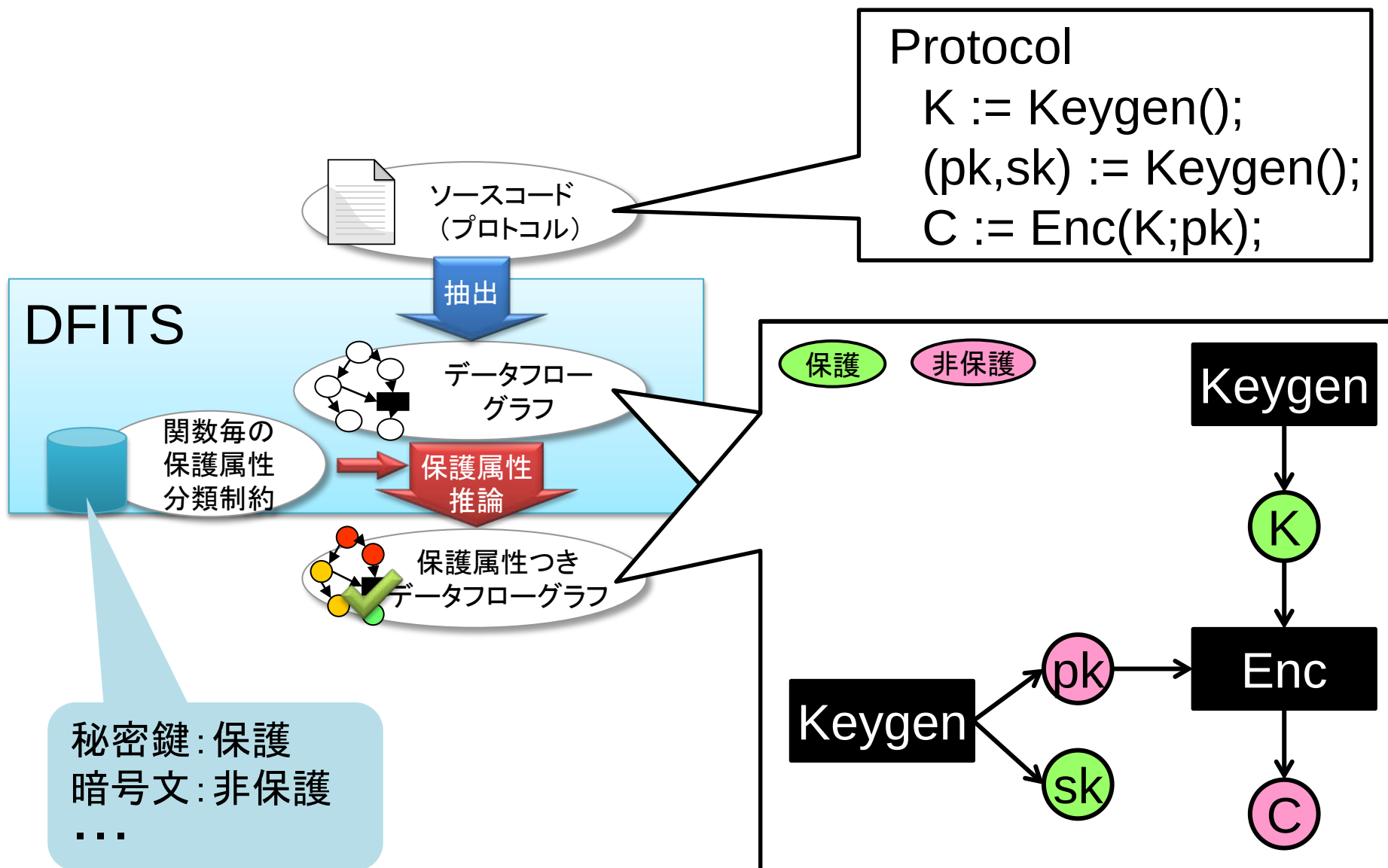
イメージ



イメージ



イメージ



保護属性(機密性限定)

• 機密性の2つの観点

- データを保護するかどうか？
- データの機密性が保証できるかどうか？

DFITS で保証したいこと:

Exposed なデータを見ても、
Confidential なデータを推測できない

		機密性保護	
		なし	あり
機密性保証	なし	Exposed	Hidden
	あり		Confidential

Exposed → Hidden

Confidential → Hidden

・機密性があるもの
・機密性がないもの
が混在
(DFITSでは分類不可)

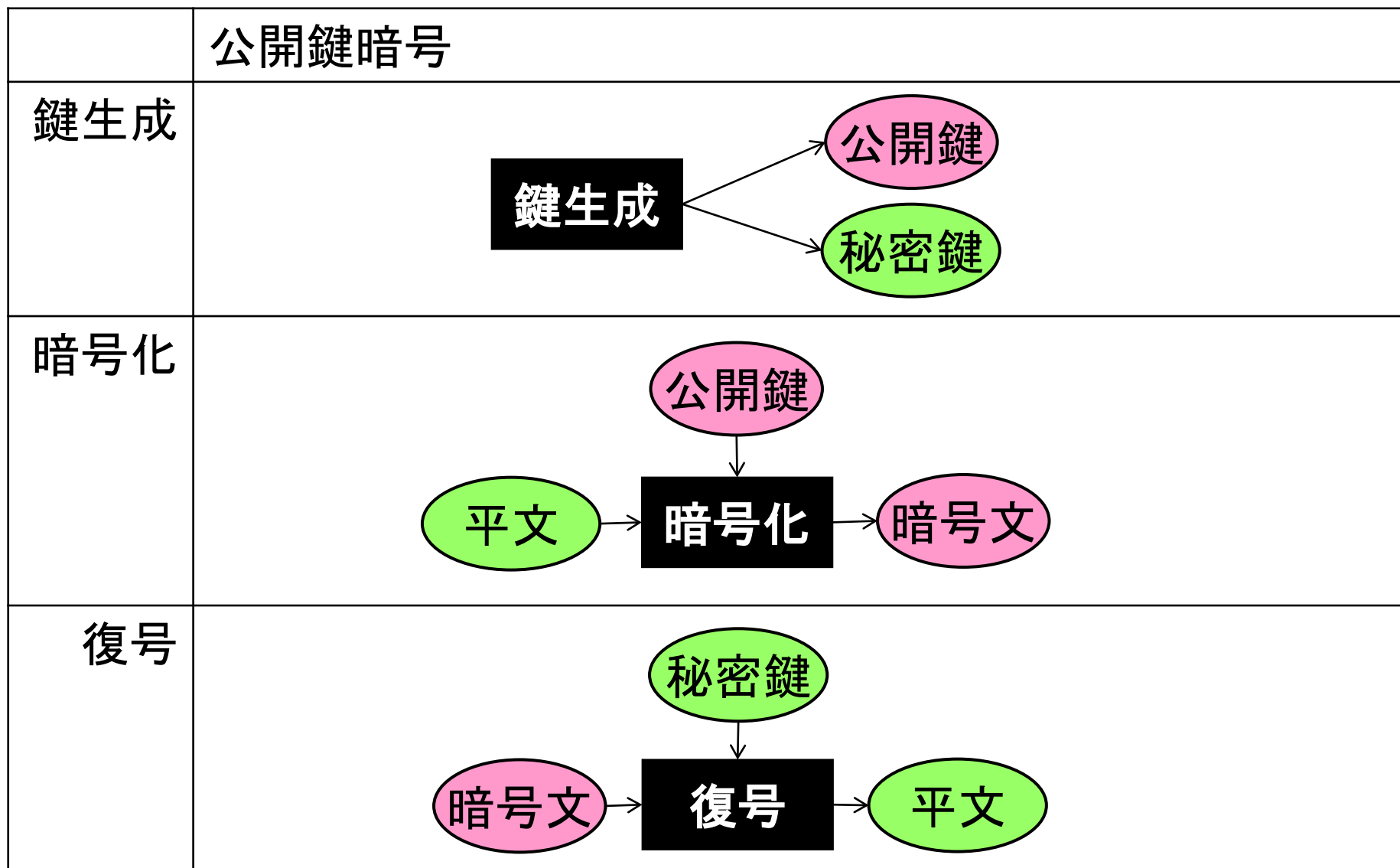
A → B : 保護属性が A の変数を, 保護属性が B の
変数に変換可能

分類規則：公開鍵暗号

Confidential
(保護・保証有)

Hidden
(保護・保証無)

Exposed
(非保護)

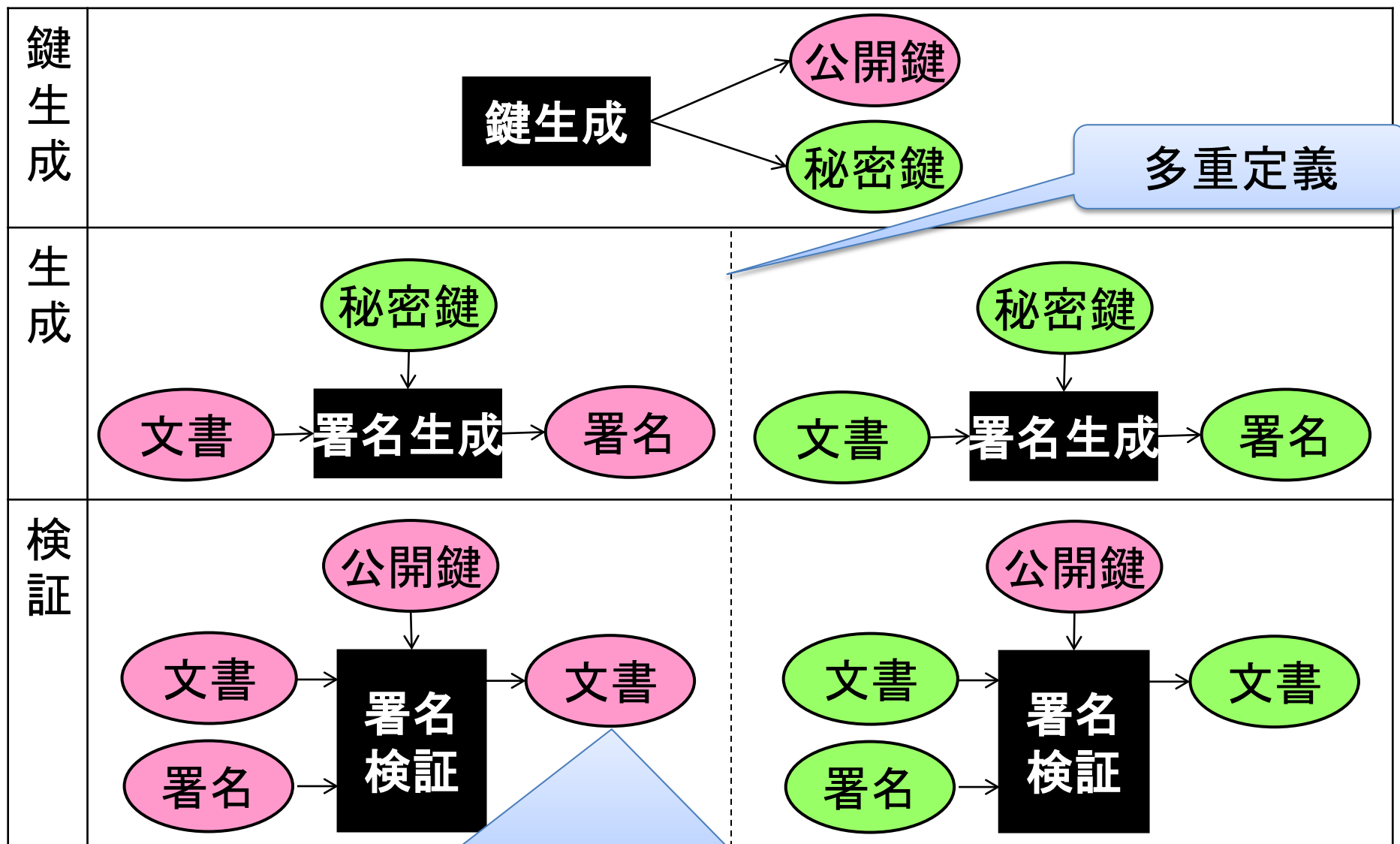


分類規則：署名

Confidential
(保護・保証有)

Hidden
(保護・保証無)

Exposed
(非保護)



多重定義

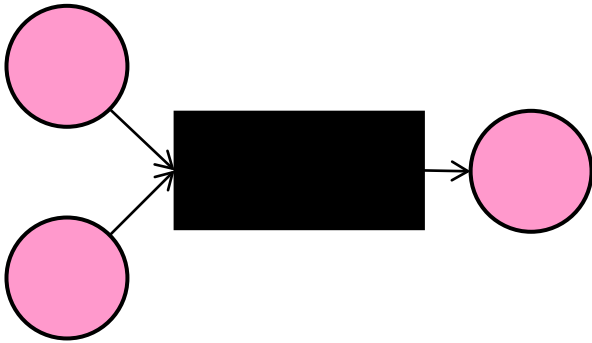
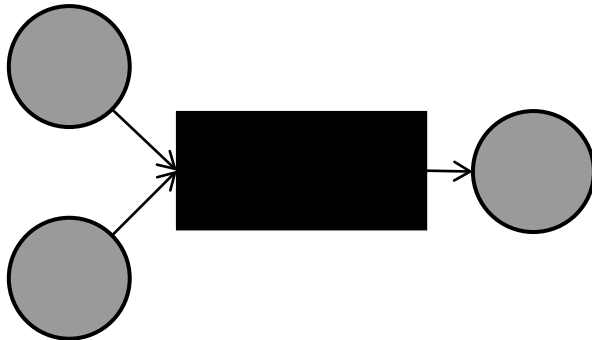
検証成功なら入力文書, 失敗なら special symbol

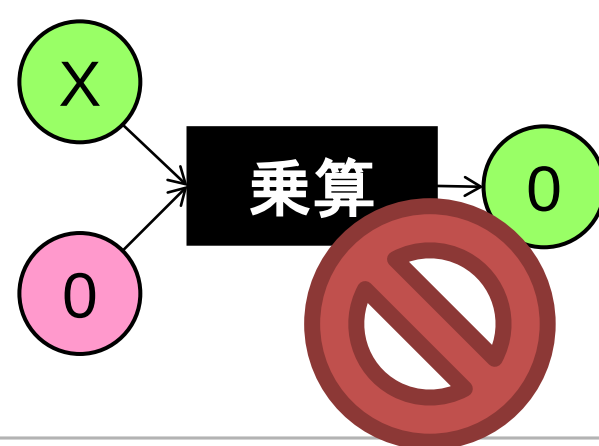
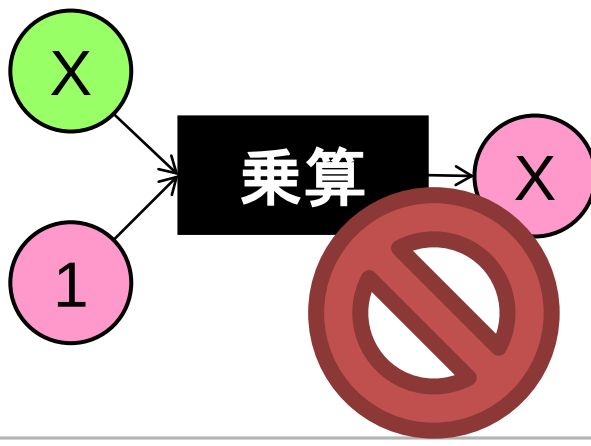
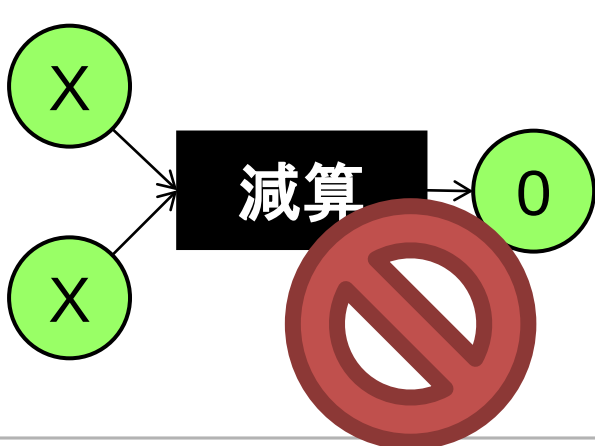
分類規則：暗号以外

Confidential
(保護・保証有)

Hidden
(保護・保証無)

Exposed
(非保護)

入出力		
四則演算等		

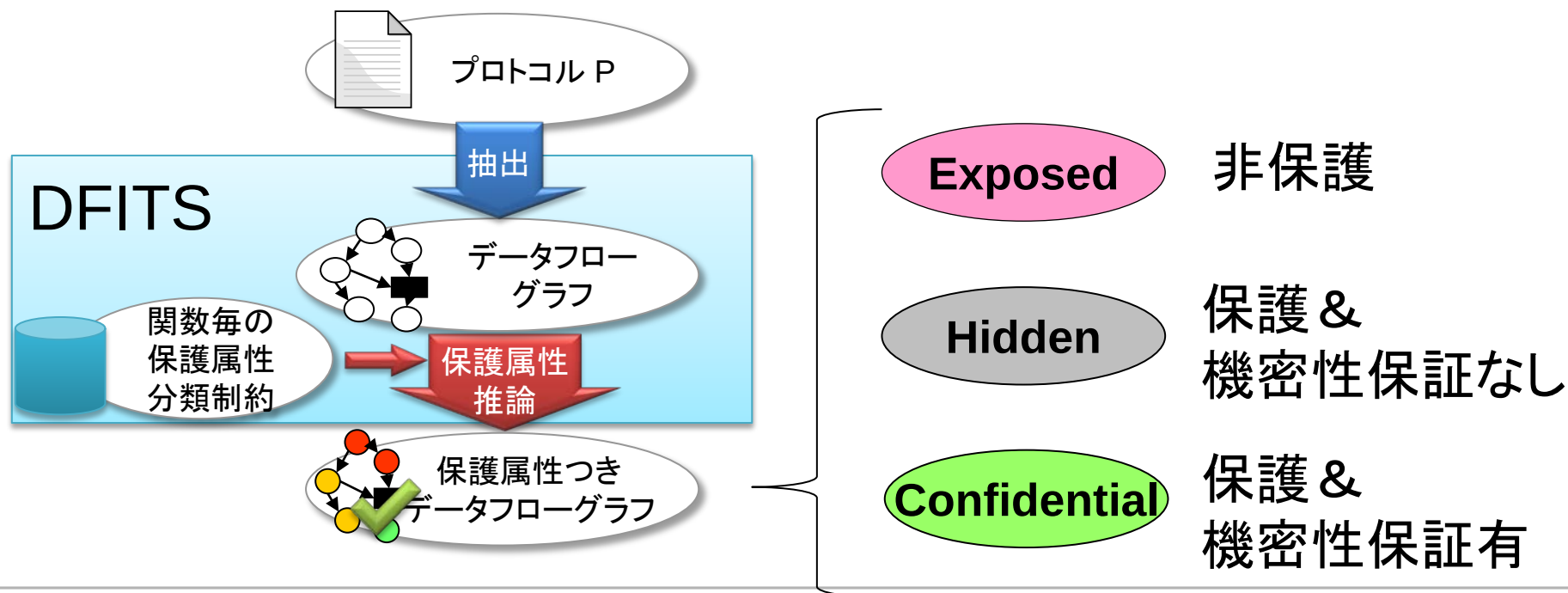


もくじ

- ソフトウェア保護技術
- DFITS (Data Flow Isolation Technology for Security)
- 機密性検証
- まとめ

定理(証明したこと)

- 仮定(後述)をみたす任意のプロトコルPと, DFITS が決定した保護属性について, 任意の攻撃者は Exposed なデータを見ても, Confidential なデータを推測できない(一方向性)
 - Hidden, Confidential が理想的に保護されているというセッティング
 - Hidden は機密性保証の対象外
 - 攻撃者は passive (データの改ざんをしない)

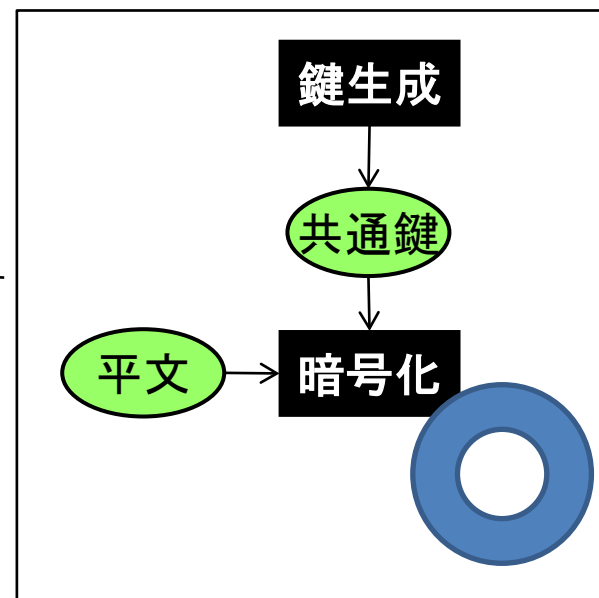


証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う
 - Key cycle がない
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない

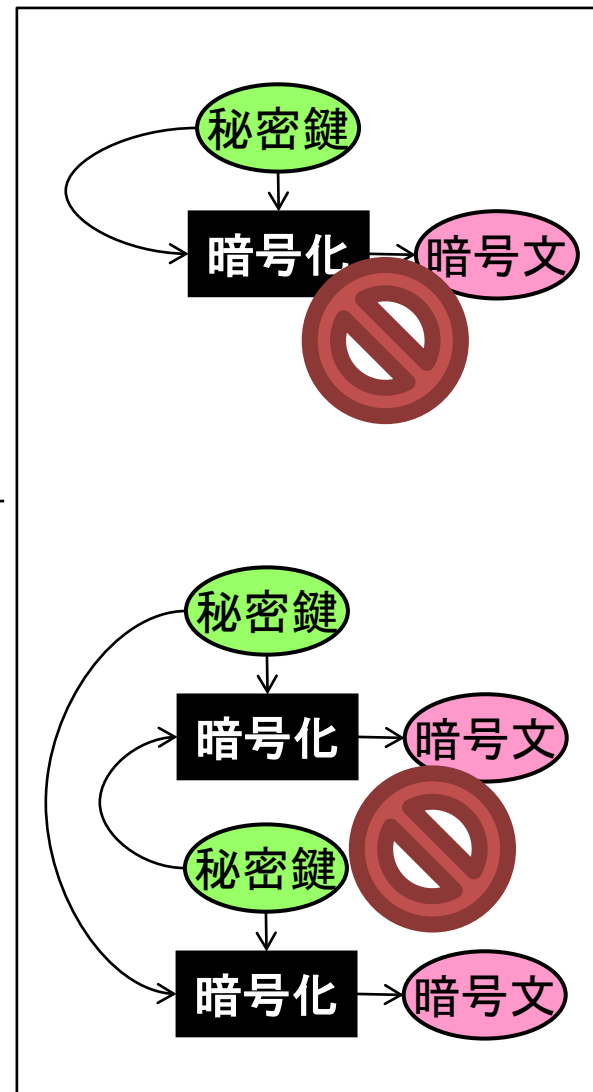
証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う ←
 - Key cycle がない
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない



証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う
 - Key cycle がない ←
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない

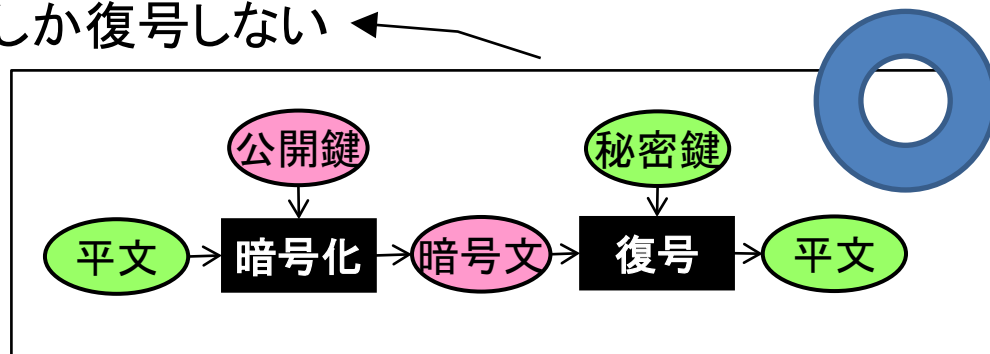


証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う
 - Key cycle がない
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない

証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う
 - Key cycle がない
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない ←

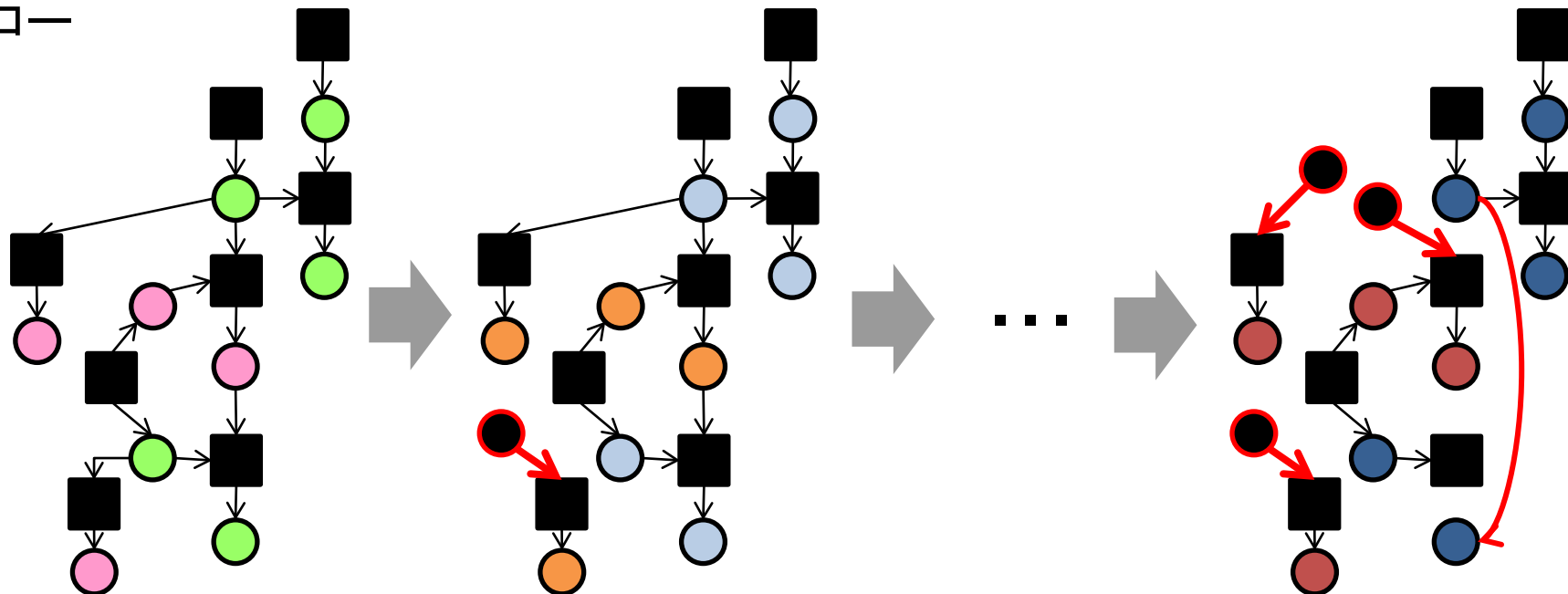


証明に用いる仮定

- 使っている暗号プリミティブの安全性
 - 暗号: 機密性 (IND-CCA)
 - 署名: 偽造不可能性 (UF-CMA)
 - ハッシュ: 理想的なハッシュ (ランダムオラクル)
- 上記安全性を保証するための仮定
 - 正しい鍵を使う
 - Key cycle がない
 - ハッシュ関数の独立性
- プロトコルの正当性を保証するための仮定
 - 暗号化関数で暗号化したものしか復号しない

証明の方法: ゲーム法

DFITS が保護属性を
決定した任意のデータ
フロー



示したいこと

○ を見ても
○ はわからない

○ を見ても
○ はわからない

○ を見ても
○ はわからない

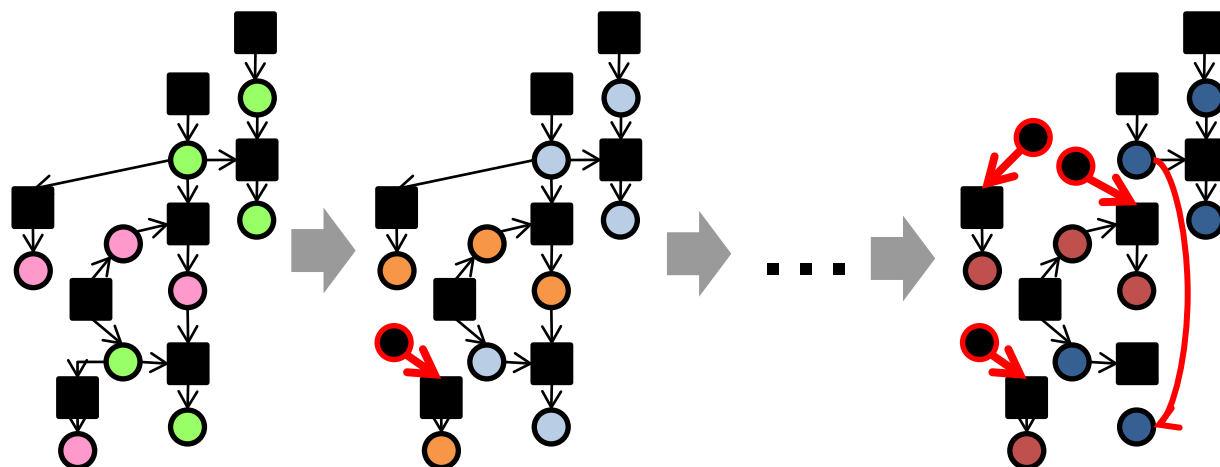
○ と ○ は
見分けがつかない

○ と ○ は
見分けがつかない



証明のポイント

- データフローをプリミティブ単位に分割
 - 秘密鍵を入力とする関数について、その入力を乱数に置き換える
 - 置き換えたとしても, exposed 変数の分布は見分けがつかない
 - Key cycle がないという仮定より, 分割が可能となる
- 分割により, 安全性が各プリミティブの安全性に帰着できる
 - プリミティブの安全性に帰着するために, Key cycle がない, 鍵を正しく使っている, ハッシュが独立などの仮定が必要

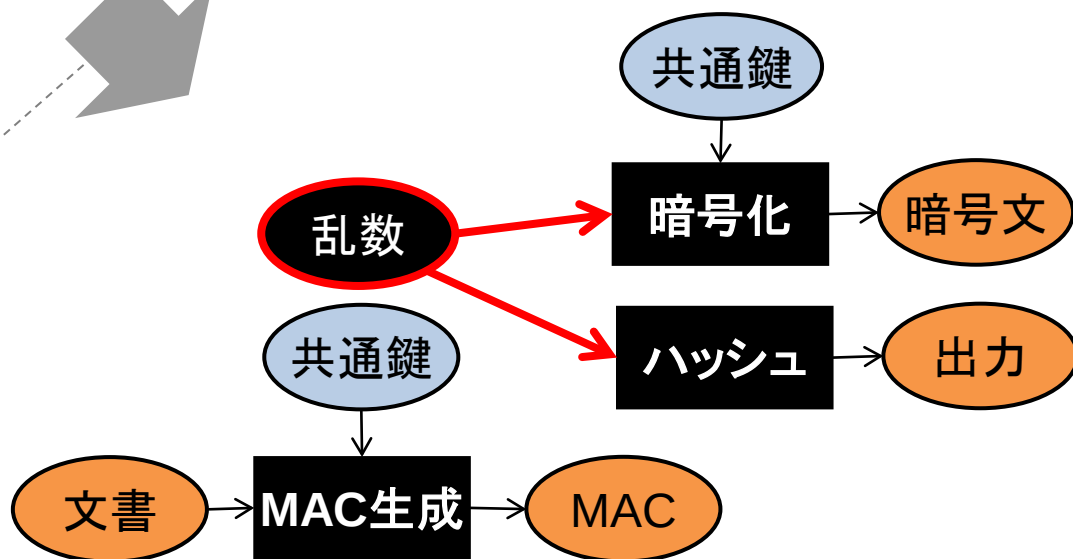


暗号プリミティブの分離

と の
見分けがつかない事を
証明できる

MAC と暗号・ハッシュが
非依存

MAC と暗号・ハッシュが
依存

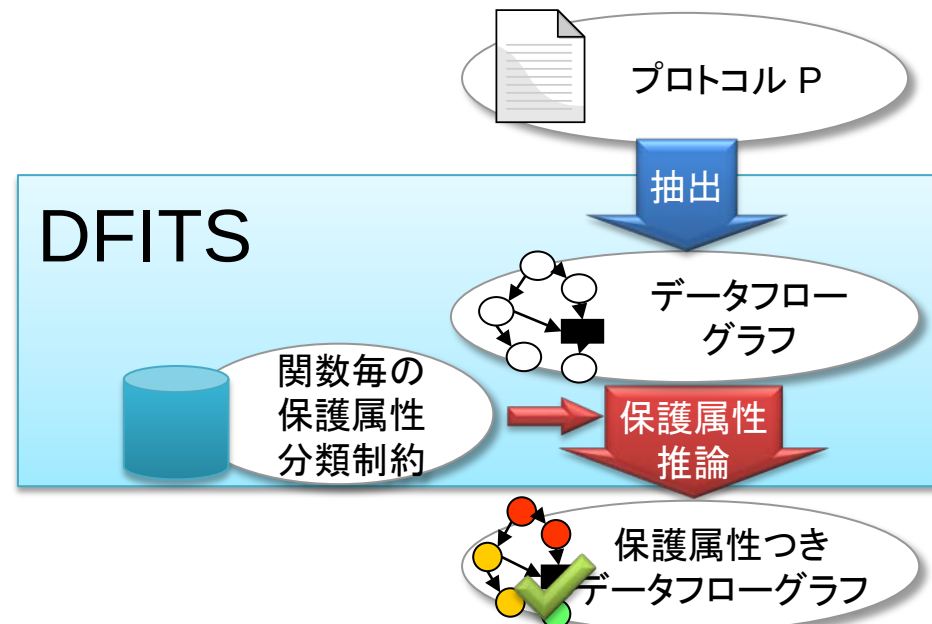


もくじ

- ソフトウェア保護技術
- DFITS (Data Flow Isolation Technology for Security)
- 機密性検証
- まとめ

まとめ

- **DFITS (Data Flow Isolation Technology for Security)**
 - セキュリティプリミティブに注目したデータフロー解析
 - データを confidential, hidden, exposed に分類
- **機密性**
 - passive な攻撃者は, exposed データを見ても, confidential データを予測できない
- **今後の課題**
 - 完全性の検討
 - より強い機密性
 - 識別不可能性



TOSHIBA

Leading Innovation >>>