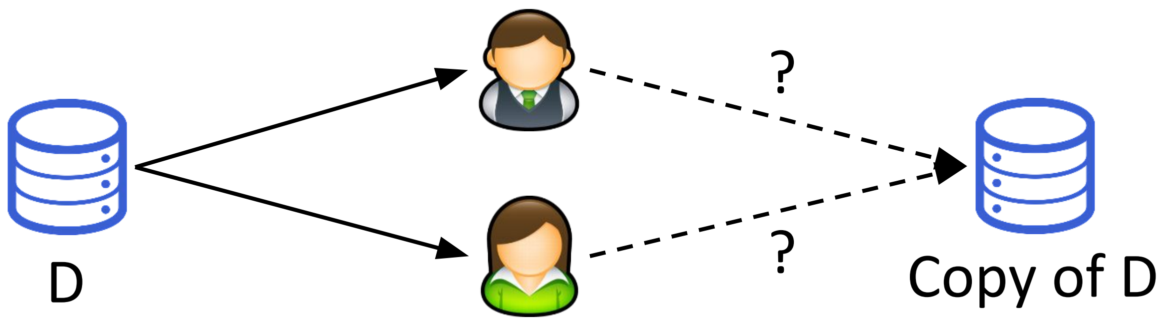
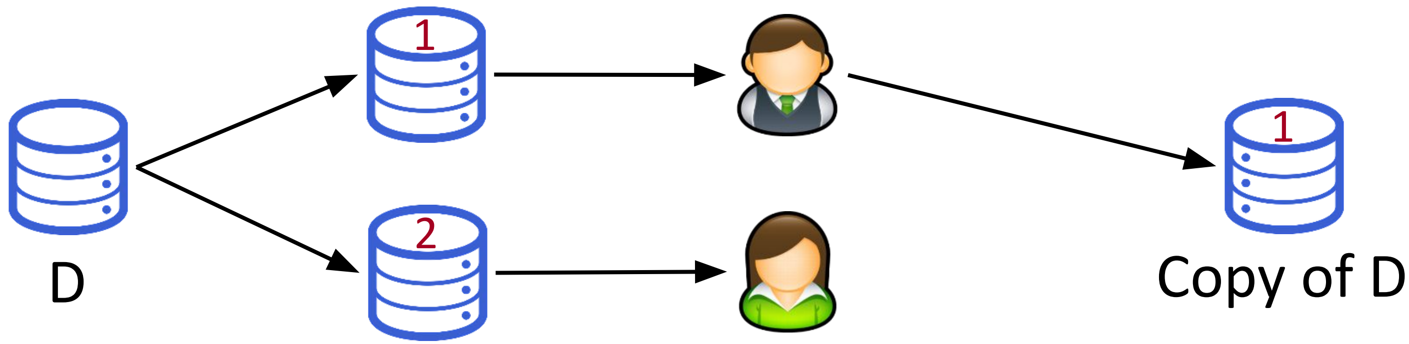
CS5322 Database Security

Watermarks: Motivation



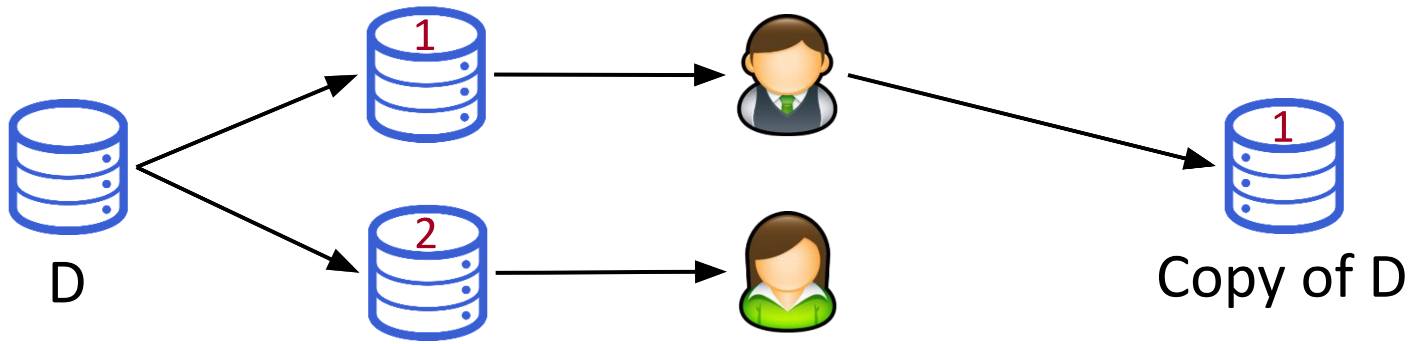
- Suppose that we have a dataset D, and we share it with two users U1 and U2
- A while later, we find that a copy of D is leaked to the internet
- Both U1 and U2 deny being the culprit
- How can we find out whom we shall blame?

Watermarks: Motivation



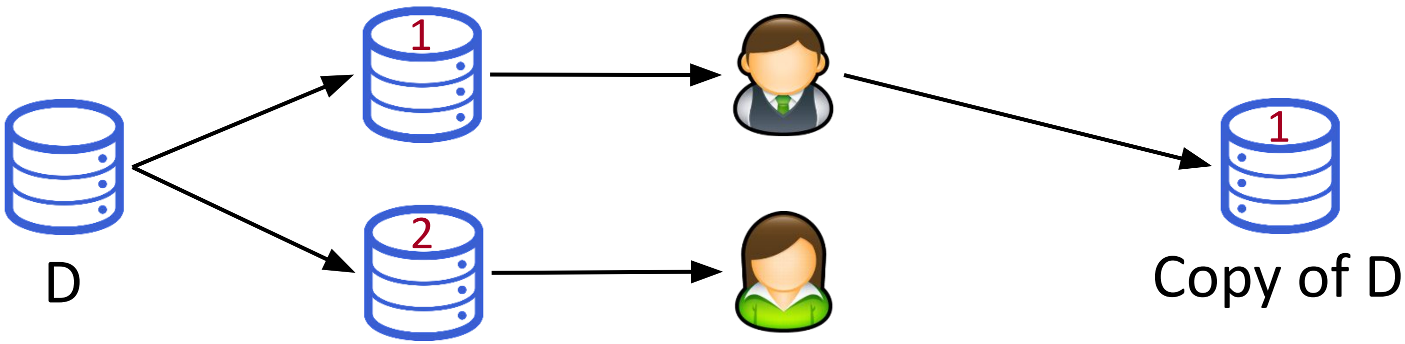
- Idea:
  - Secretly *watermark* the data shared with U1 and U2
  - If a copy of the data is leaked, we just need to check the watermark of the leaked data to decide the culprit

Watermarks: Challenges



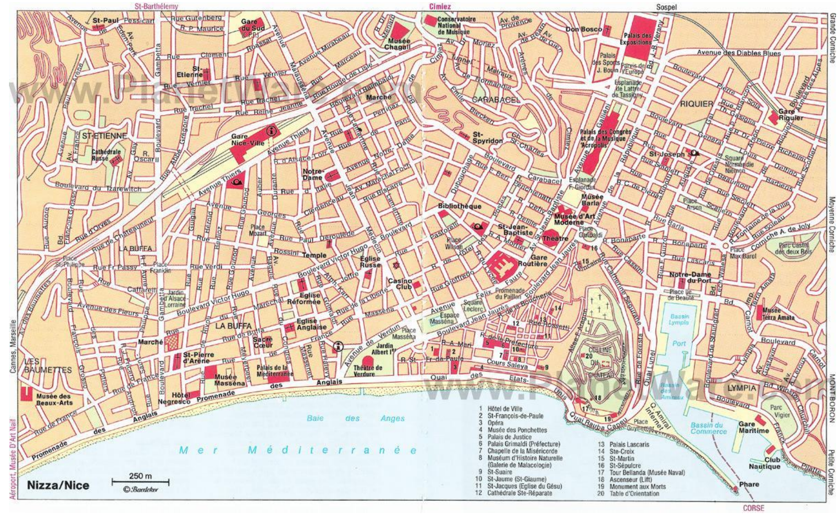
- Challenges:
  - Users may not leak the complete copy of D
    - He may choose to leak only a subset of the records
    - Or he may choose to leak only a subset of the attributes
  - Users may modify the records in D to try to destroy the watermarks

Watermarks: Challenges



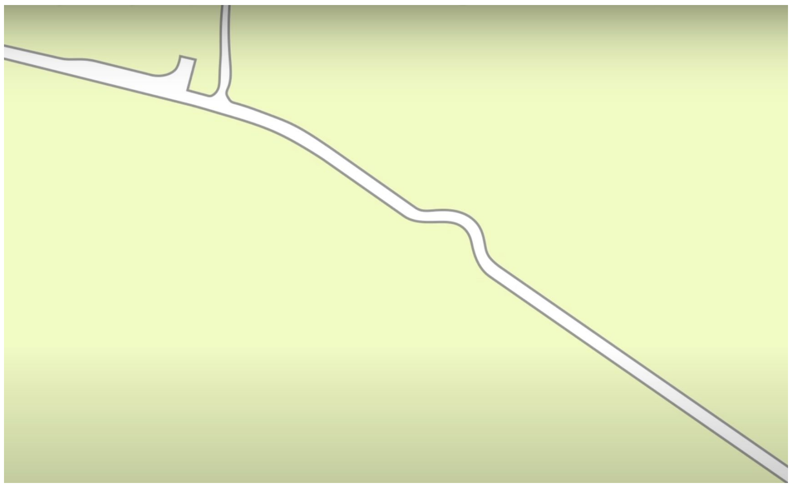
- How do we address the above challenges?
- Let's revisit the watermarking techniques for other types of data, and borrow ideas from there

Traps in Maps



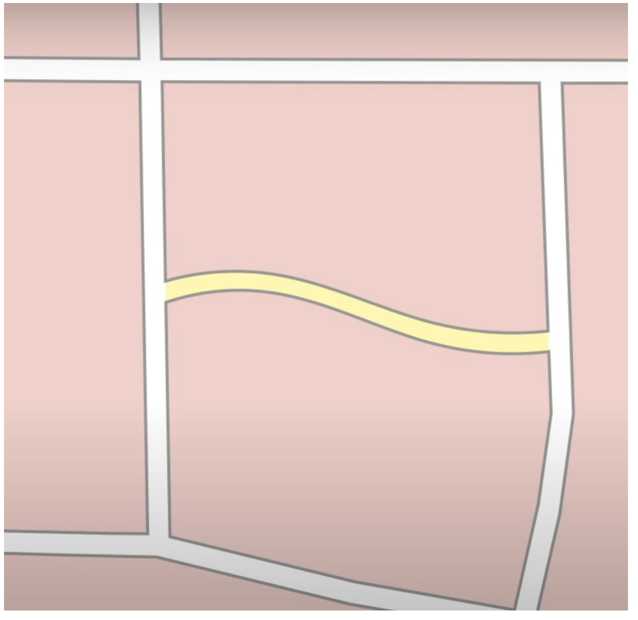
- Map making is costly
  - Especially in the old days
- To save cost, a map maker may just copy another's map
- To protect their copyrights, map makers often put *traps* in their maps
  - i.e., intentionally put incorrect information in maps
- Rationale: If the traps appear in someone else's map, then they can make a case against them

Traps in Maps



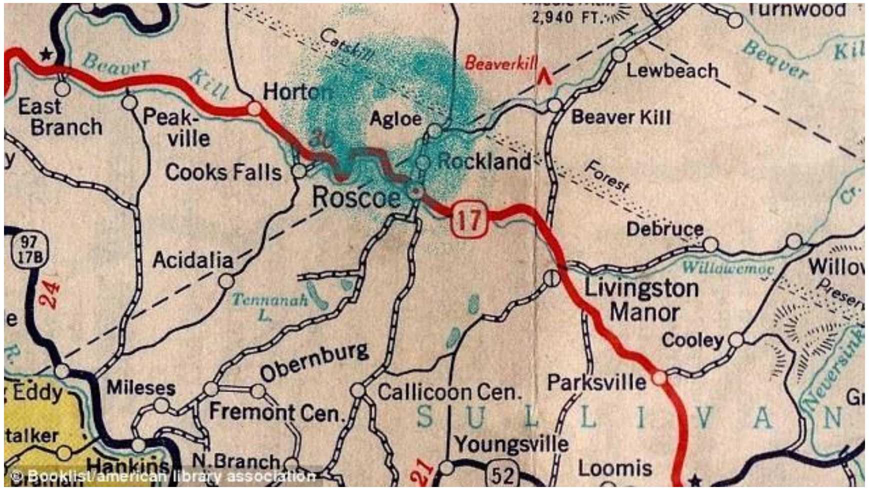
- Example:
  - Adding non-existing curves to roads

Traps in Maps



- Example:
  - Adding non-existing streets

Traps in Maps



- Example
  - Adding non-existing towns
- See
  - https://en.wikipedia.org/wiki/Agloe,_New_York
  - https://en.wikipedia.org/wiki/Argleton

Traps in Maps -> Watermarks in Data?

- Can we apply a similar approach to watermark relational data?
- Idea: Introduce false information into the data
  - Just like traps in maps
- Requirements:
  - The false information should be difficult to detect
  - And it should not affect the utility of the data

Watermarks for Relational Data

- We will discuss the approach in the following paper:
  - R. Agrawal, P. J. Hass, J. Kiernan, "Watermarking Relational Data: Framework, Algorithms and Analysis", VLDB Journal, 2003.
- We will refer to this approach as AHK
- Basic Idea:
  - Utilize the least significant bits (LSB) of the numeric values in the data

AHK: Assumptions

- The AHK approach makes the following assumptions:
  - The table T to be watermarked has a number of numeric attributes A1, A2, …, Ad, and a primary key P
  - The leaked data contains the primary key P and some numeric attributes
  - For each Ai, the ξ least significant bits (LSB) could be used for watermarking, i.e.,
    - Changes to a small number of them won't significantly affect the usefulness of the data
    - But completely removing those ξ bits would make the data much less useful
  - Detection of watermarks should not need to access T
    - Because T might be updated after being shared

AHK: Basic Idea

- For each tuple t, use its primary key t.P to decide
  - Whether the tuple should be watermarked
  - Which attribute of the tuple should be watermarked
  - Which of the ξ least significant bits should be watermarked
- But can't let users figure out which bits are changed
  - Otherwise the watermarks can be removed
- Solution
  - Pick the watermarked tuples and bits using a cryptographic pseudo-random number generator (CPRNG)

Cryptographic Pseudo-Random Number Generator (CSPNG)

- A deterministic function *G* that given a *seed value*, generates a sequence of random numbers, such that
  - Without the seed, it is computationally infeasible to predict the next random number
- How it is used:
  - The data owner chooses a secret key K
  - For each tuple t, the data owner concatenates K and its primary key t.P, and use the concatenation result K||t.P as the seed value for G
  - The data owner then uses the random numbers generated by G to decide whether and how to watermark t

AHK Watermarking Algorithm

- Input parameters:
  - G: a CPRNG
  - K: a secret key selected by the data owner
  - T: a table to be watermarked
  - d: the number of attributes in T
  - ξ: the number of least significant bits (LSB) suitable for watermarks
  - f: the fraction of tuples to be watermarked
- For each tuple t in T
  - Seed G with $K \| t.P$
  - Get random numbers $r1, r2, r3, r4$ from G
  - If ($r1$ mod $1/f$) = 0 <------------------- t has f probability to be watermarked
  - i = $r2$ mod d <------------------- the i-th attribute $t.Ai$ will be marked
  - j = $r3$ mod ξ <------------------- the j-th LSB of $t.Ai$ will be marked
  - if ($r4$ is even) then set the j-th LSB of $t.Ai$ to 0
  - else set the j-th LSB of $t.Ai$ to 1

AHK Watermarking Algorithm

- For each tuple *t* in *T*
  - Seed *G* with *K*‖*t.P*
  - Get random numbers *r1, r2, r3, r4* from *G*
  - If (*r1* mod 1/*f*) = 0 <--- *t* has *f* probability to be watermarked
  - *i* = *r2* mod *d* <--- the *i*-th attribute *t.Ai* will be marked
  - *j* = *r3* mod *ξ* <--- the *j*-th LSB of *t.Ai* will be marked
  - if (*r4* is even) then set the *j*-th LSB of *t.Ai* to 0
  - else set the *j*-th LSB of *t.Ai* to 1

- Summary:
  - Around *f* fraction of the tuples are watermarked
  - Each watermarked tuple has one watermarked bit
  - So if there are *n* tuples in *T*, there are around *n***f* watermarked bits in total

AHK Watermarking Algorithm

- For each tuple *t* in *T*
  - Seed *G* with *K*‖*t.P*
  - Get random numbers *r1, r2, r3, r4* from *G*
  - If (*r1* mod 1/*f*) = 0 <--- *t* has *f* probability to be watermarked
  - *i* = *r2* mod *d* <--- the *i*-th attribute *t.Ai* will be marked
  - *j* = *r3* mod *ξ* <--- the *j*-th LSB of *t.Ai* will be marked
  - if (*r4* is even) then set the *j*-th LSB of *t.Ai* to 0
  - else set the *j*-th LSB of *t.Ai* to 1
- Observation:
  - If the *j*-th LSB of *t.Ai* is originally 0, and *r4* happens to be even, then the watermark does not change anything
  - If the *j*-th LSB of *t.Ai* is originally 1, and *r4* happens to be odd, then the watermark does not change anything
- So how do we know check whether a bit is really watermarked?

AHK Watermark Detection Algorithm

- Input Parameters:
  - ...
  - S: a set of tuples leaked
  - τ: a threshold
- total_count = match_count = 0
- For each tuple t in S
  - Seed G with $K_u \| t.P$ <------------------- K_u is the key used to mark the data given to u
  - Get random numbers $r1, r2, r3, r4$ from G
  - If ($r1$ mod $1/f$) = 0
  - $i = r2$ mod d
  - $j = r3$ mod ξ
  - total_count = total_count + 1
  - if ($r4$ is even and the j-th LSB of $t.Ai$ is 0) or ($r4$ is odd and the j-th LSB of $t.Ai$ is 1)
  - match_count = match_count + 1
- If (match_count > total_count - τ) then suspect data leak from User u
- Else if (match_count < τ) then suspect data leak and watermark manipulation from User u

innocent data will have match_count half of total_count

- because 50% chance of match_count matching

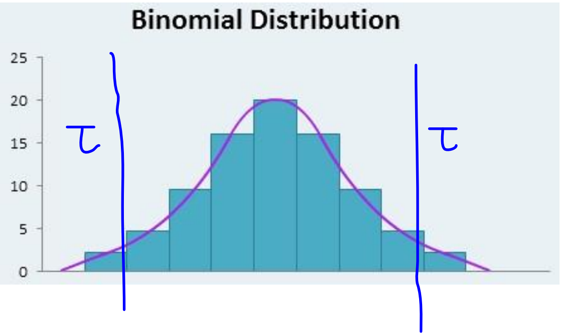
AHK Watermark Detection Algorithm

- total_count = match_count = 0
- For each tuple *t* in *S*
  - Seed *G* with $K_u \| t.P$
  - Get random numbers *r1, r2, r3, r4* from *G*
  - If (*r1* mod 1/*f*) = 0
  - *i* = *r2* mod *d*
  - *j* = *r3* mod ξ
  - total_count = total_count + 1
  - if (*r4* is even and the *j*-th LSB of *t.Ai* is 0) or (*r4* is odd and the *j*-th LSB of *t.Ai* is 1)
  - match_count = match_count + 1
- If (match_count > total_count - τ) then suspect data leak from User u
- Else if (match_count < τ) then suspect data leak and watermark manipulation from User u

- Intuition:
  - If the data is leaked by User u, then there should be a lot of match bits
  - If the data is not leaked by User u, then there should be around 50% match bits
  - If the fraction of matched bits is much smaller than 50%, then it is likely that User u has figured out which bits are watermarked , and has deliberately modified those bits

AHK Watermark Detection Algorithm

- total_count = match_count = 0
- For each tuple *t* in *S*
  - Seed *G* with $K_u \| t.P$
  - Get random numbers *r1, r2, r3, r4* from *G*
  - If (*r1* mod 1/*f*) = 0
  - *i* = *r2* mod *d*
  - *j* = *r3* mod *ξ*
  - total_count = total_count + 1
  - if (*r4* is even and the *j*-th LSB of *t.Ai* is 0) or (*r4* is odd and the *j*-th LSB of *t.Ai* is 1)
  - match_count = match_count + 1
- If (match_count > total_count - *τ*) then suspect data leak from User u
- Else if (match_count < *τ*) then suspect data leak and watermark manipulation from User u



- How should we decide *τ*?
- If the data is not leaked by User u, then each "marked" bit has 1/2 probability to produce a match
- When we have *total_count* marked tuples, the number of matched bits should follow a binomial distribution B(*total_count*, 1/2)
- We can set *τ* such that the probability of getting a binomial variable larger than *total_count* – *τ* or smaller than *τ* is small (e.g., 5%)

AHK Watermark Detection Algorithm

- total_count = match_count = 0
- For each tuple t in S
  - Seed G with $K_u \| t.P$
  - Get random numbers $r1, r2, r3, r4$ from G
  - If ($r1$ mod $1/f$) = 0
  - $i = r2$ mod d
  - $j = r3$ mod ξ
  - total_count = total_count + 1
  - if ($r4$ is even and the j-th LSB of $t.Ai$ is 0) or ($r4$ is odd and the j-th LSB of $t.Ai$ is 1)
  - match_count = match_count + 1
- If (match_count > total_count - τ) then suspect data leak from User u
- Else if (match_count < τ) then suspect data leak and watermark manipulation from User u

- Note that the above algorithm can be easily extended to handle the case when S contains only a subset of the attributes
- How?
- After choosing i, we only increase total_count when Ai exists in S

Exercise

- For each tuple *t* in *T*
  - Seed *G* with *K*‖*t.P*
  - Get random numbers *r1, r2, r3, r4* from *G*
  - If (*r1* mod 1/*f*) = 0 <--- *t* has 1/*f* probability to be watermarked
  - *i* = *r2* mod *d* <--- the *i*-th attribute *t.Ai* will be marked
  - *j* = *r3* mod *ξ* <--- the *j*-th LSB of *t.Ai* will be marked
  - if (*r4* mod 3 = 1) then set the *j*-th LSB of *t.Ai* to 0
  - else set the *j*-th LSB of *t.Ai* to 1

- Suppose that we apply the above watermarking algorithm on a table *T*
- Explain whether and/or how we can detect watermarks from the modified table without accessing *T*

Exercise

- Input Parameters:
  - ...
  - S: a set of tuples leaked
  - τ: a threshold
- total_count = match_count = 0
- For each tuple t in S
  - Seed G with $K_u \| t.P$
  - Get random numbers $r1, r2, r3, r4$ from G
  - If ($r1$ mod $1/f$) = 0
  - i = $r2$ mod d
  - j = $r3$ mod ξ
  - total_count = total_count + 1
  - if ($r4$ mod 3 = 1 and the j-th LSB of $t.Ai$ is 0) or ($r4$ mod 3 != 1 and the j-th LSB of $t.Ai$ is 1)
  - match_count = match_count + 1
- If (match_count > total_count – $\tau 1$) then suspect data leak from User u
- Else if (match_count < $\tau 2$) then suspect data leak and watermark manipulation from User u

Exercise

- total_count = match_count = 0
- For each tuple t in S
  - Seed G with $K_u \| t.P$
  - Get random numbers $r1, r2, r3, r4$ from G
  - If ($r1$ mod $1/f$) = 0
  - i = $r2$ mod d
  - j = $r3$ mod ξ
  - total_count = total_count + 1
  - if ($r4$ mod 3 = 1 and the j-th LSB of $t.Ai$ is 0) or ($r4$ mod 3 != 1 and the j-th LSB of $t.Ai$ is 1)
  - match_count = match_count + 1
- If (match_count > total_count – $\tau 1$) then suspect data leak from User u
- Else if (match_count < $\tau 2$) then suspect data leak and watermark manipulation from User u

- How should we decide $\tau 1$ and $\tau 2$?
- If the data is not leaked by User u, then each "marked" bit has 1/3 ~ 2/3 probability to produce a match
- When we have *total_count* marked tuples, the number of matched bits should follow a binomial distribution B(*total_count*, *p*), where p is in [1/3, 2/3]
- We can set $\tau 1$ and $\tau 2$ such that the probability of getting a binomial variable larger than *total_count* – $\tau 1$ or smaller than $\tau 2$ is small (e.g., 5%)

Exercise

- For each tuple *t* in *T*
  - Seed *G* with *K*‖*t.P*
  - Get random numbers *r1, r2, r3* from *G*
  - If (*r1* mod 1/*f*) = 0 <--- *t* has 1/*f* probability to be watermarked
  - *i* = *r2* mod *d* <--- the *i*-th attribute *t.Ai* will be marked
  - *j* = *r3* mod *ξ* <--- the *j*-th LSB of *t.Ai* will be marked
  - Flip the *j*-th LSB of *t.Ai*

- Suppose that we apply the above watermarking algorithm on a table *T*
- Explain whether and/or how we can detect watermarks from the modified table without accessing *T*

Exercise

- For each tuple *t* in *T*
  - Seed *G* with *K*‖*t.P*
  - Get random numbers *r1, r2, r3* from *G*
  - If (*r1* mod 1/*f*) = 0 <--- *t* has 1/*f* probability to be watermarked
  - *i* = *r2* mod *d* <--- the *i*-th attribute *t.Ai* will be marked
  - *j* = *r3* mod *ξ* <--- the *j*-th LSB of *t.Ai* will be marked
  - Flip the *j*-th LSB of *t.Ai*

- Suppose that we apply the above watermarking algorithm on a table *T*
- Explain whether and/or how we can detect watermarks from the modified table ~~without accessing *T*~~

Exercise

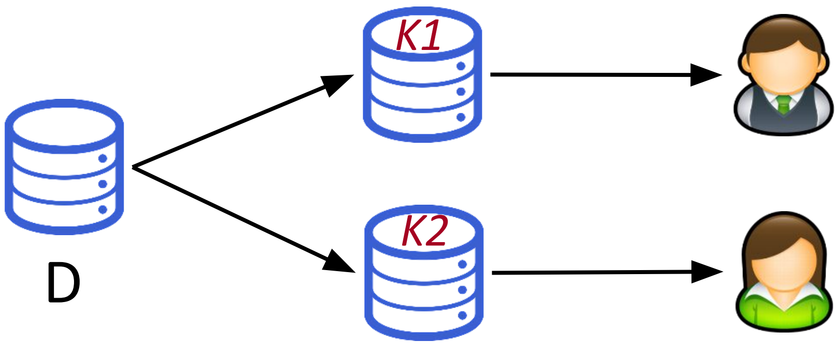
- Input Parameters:
  - *. . .*
  - *S*: a set of tuples leaked
  - *τ*: a threshold
- total_count = match_count = 0
- For each tuple *t* in *S*
  - Seed *G* with $K_u \| t.P$
  - Get random numbers *r1, r2, r3* from *G*
  - If (*r1* mod 1/*f*) = 0
  - *i* = *r2* mod *d*
  - *j* = *r3* mod *ξ*
  - total_count = total_count + 1
  - if (the *j*-th LSB of *t.Ai* is different from the original tuple)
  - match_count = match_count + 1
- If (match_count > *τ1*) then suspect data leak from User u

Exercise

- total_count = match_count = 0
- For each tuple *t* in *S*
  - Seed *G* with $K_u \| t.P$
  - Get random numbers *r1, r2, r3* from *G*
  - If (*r1* mod 1/*f*) = 0
  - *i* = *r2* mod *d*
  - *j* = *r3* mod *ξ*
  - total_count = total_count + 1
  - if (the *j*-th LSB of *t.Ai* is different from the original tuple)
  - match_count = match_count + 1
- $\boxed{\text{If (match\_count > } \tau 1\text{) then suspect data leak from User u}}$

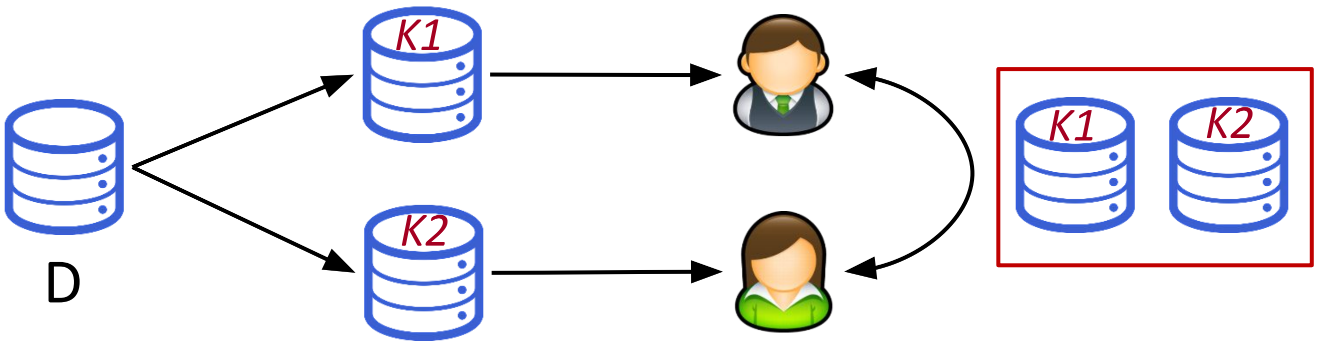
- How should we decide *τ1*?
- If the data is not leaked by User u but another User v, then each "marked" bit has *f*/d/*ξ* probability to produce a match
- When we have *total_count* marked tuples, the number of matched bits should follow a binomial distribution B(*total_count*, *f*/d/*ξ*))
- We can set *τ1* such that the probability of getting a binomial variable larger than *τ1* is small (e.g., 5%)

Attacking AHK Watermarks



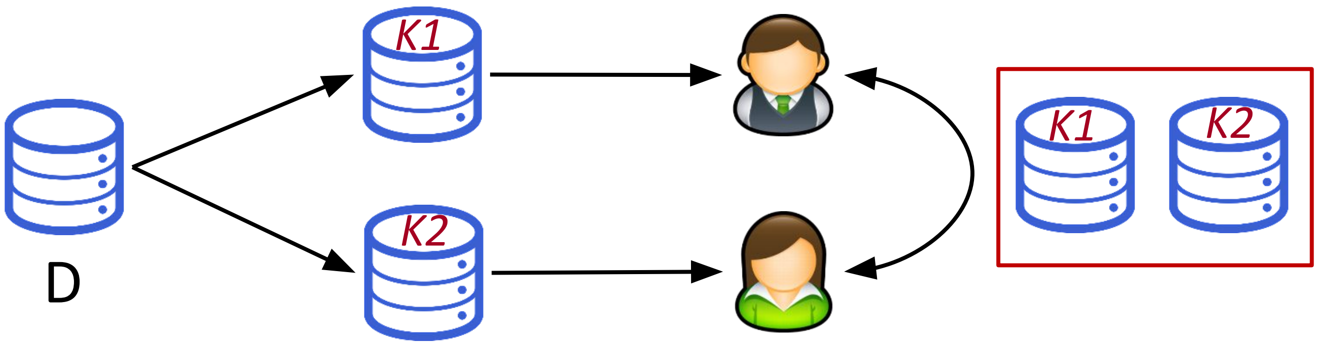
- Summary:
  - Choose a secret key *Ku* for each user u
  - Use *Ku* to decide which bit of which tuple should be watermarked
- Question:
  - If you are one of the users, what would you do to attack the watermarks?

Attacking AHK Watermarks



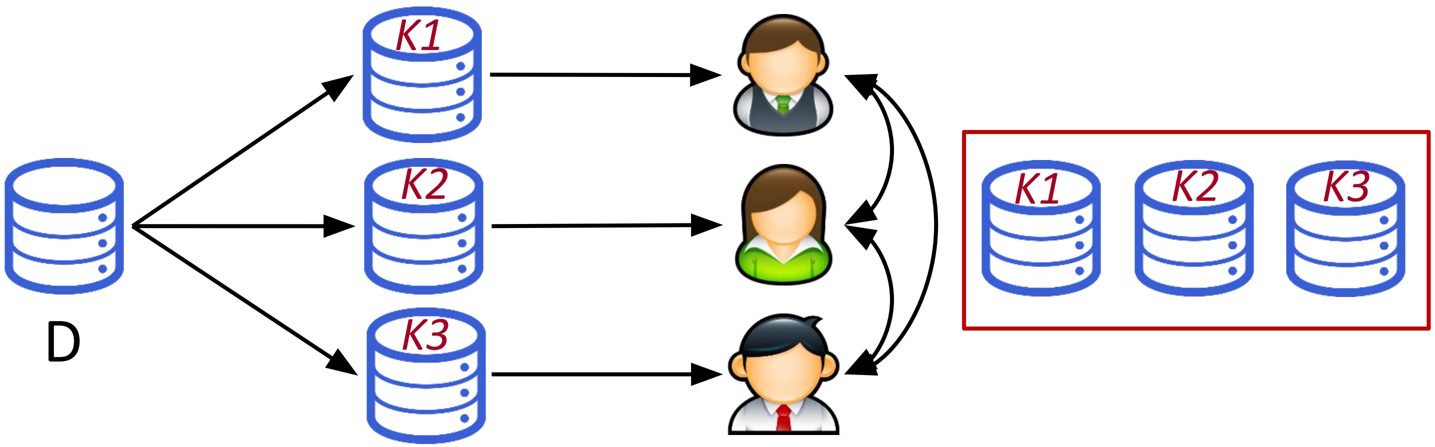
- Idea:
  - Collude with another user to obtain two differently watermarked datasets
  - Compare the two datasets to figure out which bits are different
- Rationale:
  - In each dataset, the watermarked tuples and watermarked bits are all randomly selected
  - The chance that the same bit is watermarked in both dataset is fairly low
  - So a watermarked bit in one dataset is likely to be different in the other dataset

Attacking AHK Watermarks



- Problem:
  - We only know which bits are different in the two datasets
  - But we don’t know whether a "different" bit is watermarked in Dataset 1 or Dataset 2
- Solution 1:
  - We don’t care; just flip half of those bits
  - This could fool the watermark detection algorithm described previously

Attacking AHK Watermarks



- Solution 2:
  - Find a third user to collude with
  - Now whenever there is a bit that the datasets do not agree on, we can use a majority vote to infer the original value
- Question:
  - How do we guard against such collusion attacks?
- Idea:
  - Make the watermark bits *correlated* among the datasets

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- Suppose that we are to give a dataset *D* to four users *u1*, *u2*, *u3*, and *u4*
- We randomly divide *D* into three parts: *D1*, *D2*, *D3*
- We choose a secret key *K*, and produce two watermarked versions of *Di*
  - The first version *Di+* is generated by the AHK algorithm
  - The second version *Di-* is generated by flipping the watermarked bits in *Di+*
- We then assign watermarked parts to users as shown above

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- Suppose that *u1* and *u3* collude
- They could compromise the watermarks in *D1* and *D2*
- But the watermarks in *D3* still exists
- When the data is leaked, if we detect watermarks in *D3* (but not in the other parts),
- then we know that *u1* and *u3* must be among the culprits

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- Suppose that *u1*, *u2*, and *u3* collude
- They could compromise the watermarks in *D1* and *D2*
- But the watermarks in *D3* still exist
- When the data is leaked, if we find that the watermarks exist in *D3* (but not in the other parts)
- then we know that *u1* and *u3* be among the culprits
- Why can't we be sure about *u2*?
  - Because *u1* and *u3* can already compromise *D1* and *D2*

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- Suppose that *u1* and *u4* collude
- They could compromise the watermarks in *D1* and *D2 and D3*
- When the data is leaked, if we cannot detect any watermarks
- then we know that *u1* and *u4* must be among the culprits
  - Otherwise, the watermarks in *D1*, *D2*, and *D3* cannot be compromised at the same time

Guarding Against Collusion Attacks

u1		*D1+*,	*D2+*,	*D3+*
u2		*D1-*,	*D2+*,	*D3+*
u3		*D1-*,	*D2-*,	*D3+*
u4		*D1-*,	*D2-*,	*D3-*

- Suppose that *u1*, *u2*, and *u4* collude
- They could compromise the watermarks in *D1* and *D2 and D3*
- When the data is leaked, if we cannot detect any watermarks
- then we know that *u1* and *u4* must be among the culprits
  - Otherwise, the watermarks in *D1*, *D2*, and *D3* cannot be compromised at the same time
- But we cannot be sure about *u2*

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- Suppose that *u1*, *u2*, and *u4* collude
- They could compromise the watermarks in *D1* and *D2 and D3*
- But they could be more strategic
- Can they leak D1-, D2-, D3+, and pretend that the leak is caused by *u3*?
- No.
- Because they don’t know which tuples are from D2- and D3+

Guarding Against Collusion Attacks

u1	*D1+*,	*D2+*,	*D3+*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1-*,	*D2-*,	*D3-*

- In general, if
  - there are *n* users, and
  - we assume that at most *c* of them collude
- Then
  - there are ways to distribute watermarked parts to users to ensure successful detection under our assumption
- See
  - D. Boneh and J. Shaw. "Collusion-Secure Fingerprinting for Digital Data". IEEE Transaction on Information Theory, 1998.

Extensions to Categorical Attributes

- For categorical attributes, there is no least significant bits that we can exploit
- Solution:
  - When we decide to watermark a categorical value, we change it to another value (instead of just changing a bit)
  - Example: changing the education of a person from "Bachelor" to "High School"
- Challenge:
  - Could be tricky to maintain the validity of the data
    - E.g., change a male patient’s disease from dyspepsia to breast cancer

Exercise 1

u1		*D1-*,	*D2-*,	*D3-*
u2		*D1-*,	*D2+*,	*D3+*
u3		*D1-*,	*D2-*,	*D3+*
u4		*D1+*,	*D2-*,	*D3+*

- Suppose that we are to give a dataset *D* to four users *u1*, *u2*, *u3*, and *u4*
- We randomly divide *D* into three parts: *D1*, *D2*, *D3*
- We choose a secret key *K*, and watermark each part *Di* **twice**
  - The first version *Di+* is generated the AHK algorithm
  - The second version *Di-* is generated by flipping the watermarked bits in *Di+*
- We then assign watermarked parts to users as shown above
- Analyze the security guarantee of this watermarking scheme

Exercise 1: Answer

u1	*D1-*,	*D2-*,	*D3-*
u2	*D1-*,	*D2+*,	*D3+*
u3	*D1-*,	*D2-*,	*D3+*
u4	*D1+*,	*D2-*,	*D3-*

- This watermarking scheme is not good because
  - If u1, u2, and u4 collude and use a majority vote, they can obtain a copy of the data that corresponds to {*D1-*, *D2-*, *D3+*}, which is the same as u3's copy
  - By leaking this copy, u1, u2, u4 can frame u3 as the data leaker

Exercise 2

u1	*D1+*,	*D2-*,	*D3+*,	*D4-*
u2	*D1-*,	*D2+*,	*D3-*,	*D4-*
u3	*D1-*,	*D2-*,	*D3+*,	*D4+*
u4	*D1-*,	*D2+*,	*D3-*,	*D4+*

- Suppose that we are to give a dataset *D* to four users *u1*, *u2*, *u3*, and *u4*
- We randomly divide *D* into four parts: *D1*, *D2*, *D3*, *D4*
- We choose a secret key *K*, and watermark each part *Di* **twice**
  - The first version *Di+* is generated the AHK algorithm
  - The second version *Di-* is generated by flipping the watermarked bits in *Di+*
- We then assign watermarked parts to users as shown above
- Analyze the security guarantee of this watermarking scheme

Exercise 2: Answer

u1	*D1+,*	*D2-,*	*D3+,*	*D4-*
u2	*D1-,*	*D2+,*	*D3-,*	*D4-*
u3	*D1-,*	*D2-,*	*D3+,*	*D4+*
u4	*D1-,*	*D2+,*	*D3-,*	*D4+*

- This watermarking scheme is OK, because:
  - No 3-user-collusion can frame the remaining user, regardless of whether they follow the majority vote or the minority vote
  - If the users collude to destroy watermarks and they involve u1, then the watermarks in D1 will be destroyed; in that case, u1 will get caught, since he is the only user who has D1+
  - Every user collusion that does not involve u1 would destroy a different set of watermarks, and hence, we can identify the colluding users by observing which watermarks are destroyed

Exercise 3

u1	*D1+*,	*D2-*,	*D3-*,	*D4-*
u2	*D1-*,	*D2+*,	*D3-*,	*D4-*
u3	*D1-*,	*D2-*,	*D3+*,	*D4-*
u4	*D1-*,	*D2-*,	*D3-*,	*D4+*

- Suppose that we are to give a dataset *D* to four users *u1*, *u2*, *u3*, and *u4*
- We randomly divide *D* into four parts: *D1*, *D2*, *D3*, *D4*
- We choose a secret key *K*, and watermark each part *Di* **twice**
  - The first version *Di+* is generated the AHK algorithm
  - The second version *Di-* is generated by flipping the watermarked bits in *Di+*
- We then assign watermarked parts to users as shown above
- Analyze the security guarantee of this watermarking scheme

Exercise 3: Answer

u1		*D1+*,	*D2-*,	*D3-*,	*D4-*	
u2		*D1-*,	*D2+*,	*D3-*,	*D4-*	
u3		*D1-*,	*D2-*,	*D3+*,		*D4-*
u4		*D1-*,	*D2-*,	*D3-*,	*D4+*	

- This watermarking scheme is not OK, because
  - If any 3 users collude and follow the majority rule, they can leak {*D1-*, *D2-*, *D3-*, *D4-*}
  - When the data owner sees {*D1-*, *D2-*, *D3-*, *D4-*}, she would be unable to pinpoint any of the culprits, because every user can claim that it was the other 3 users who leaked the data