Passkey Discoverable Credentials

Backend Implementation Changes

Date: 2026-01-30

PROBLEM STATEMENT

Previously, passkey login required email BEFORE authentication:

- User had to enter email first
- Then click Sign in with Passkey
- This BLOCKED Google Password Manager passkey selector

Problem: allowCredentials was being sent, preventing the OS from showing all available passkeys

NEW FLOW (Snapchat-like UX):

- User clicks Sign in with Passkey (NO email needed)
- Google Password Manager shows ALL passkeys for this app
- User selects account, scans fingerprint
- Backend looks up user by credential_id
- User is logged in!

Solution: Remove allowCredentials, use credential-first lookup

NEW LIBRARY IMPORTS

Before

```
from webauthn.helpers import parse_registration_credential_json
```

After (NEW import highlighted)

```
from webauthn.helpers import (
    parse_registration_credential_json,
    parse_authentication_credential_json  # <-- NEW!
)
```

NEW HELPER FUNCTIONS ADDED

1. Global Challenge Storage (Thread-safe)

For discoverable credentials, we store the challenge globally (not per-user) since we dont know the users email yet.

```
_global_challenge_lock = threading.Lock()
_global_challenge_data = {"challenge": None, "expiry": None}

def store_global_webauthn_challenge(challenge_str, expiry_minutes=5):
    """Store challenge globally for discoverable credentials"""

def get_global_webauthn_challenge() -> Optional[str]:
    """Retrieve the global WebAuthn challenge"""

def clear_global_webauthn_challenge():
    """Clear the global WebAuthn challenge after use"""
```

2. Credential Lookup by ID

Search ALL users passkey_credentials JSON array to find the matching credential.

```
def get_credential_by_id(credential_id: str) -> Optional[dict]:
    """Find a passkey credential by its ID across ALL users"""
    # Queries: SELECT id, email, passkey_credentials
    #          FROM slay_users WHERE passkey_credentials IS NOT NULL
    # Returns: {credential_id, public_key, sign_count, user_id, user_email}
```

3. User Lookup by ID

```
def get_user_by_id(user_id: str) -> Optional[dict]:
    """Get user by their database ID"""
```

PASSKEY_LOGIN_BEGIN - CHANGES

ORIGINAL CODE (Required Email)

```
@app.route("/api/passkey/login/begin", methods=["POST"])
def passkey_login_begin():
    email = data.get("email")

    if not email:
        return error("Email is required")  # BLOCKED!

    # Get users specific credentials
    credentials = get_passkey_credentials(email)

    # Build allowCredentials (PREVENTS passkey selector!)
    allowed_credentials = []
    for cred in credentials:
        allowed_credentials.append(
            PublicKeyCredentialDescriptor(id=cred_id)
        )

    authentication_options = generate_authentication_options(
        allow_credentials=allowed_credentials,  # BLOCKS!
    )

    store_webauthn_challenge(email, challenge)  # Per-user
```

NEW CODE (No Email Required)

```
@app.route("/api/passkey/login/begin", methods=["POST"])
def passkey_login_begin():
    # Generate challenge
    challenge_str = generate_challenge()

    authentication_options = generate_authentication_options(
        rp_id=RP_ID,
        challenge=challenge_bytes,
        user_verification=UserVerificationRequirement.REQUIRED,
        # NO allowCredentials = Passkey selector shows ALL!
    )

    # Store challenge GLOBALLY (not per-email)
    store_global_webauthn_challenge(challenge_str)

    return {"success": True, "options": options_dict}
```

KEY CHANGES: No email required | No allowCredentials | Global challenge storage

PASSKEY_LOGIN_COMPLETE - CHANGES

ORIGINAL CODE (Required Email)

```
@app.route("/api/passkey/login/complete", methods=["POST"])
def passkey_login_complete():
    email = data.get("email")

    if not email:
        return error("Email is required")  # BLOCKED!

    # Get user by email FIRST
    user = get_user_by_email(email)

    # Find credential in users list
    credentials = get_passkey_credentials(email)
    for cred in credentials:
        if cred["credential_id"] == credential_id:
            matching_cred = cred

    # Get per-user challenge
    stored_challenge = user.get("verification_code")

    # Parse with Pydantic (version issues!)
    credential = AuthenticationCredential.parse_raw(json_data)
```

NEW CODE (Credential-First Lookup)

```
@app.route("/api/passkey/login/complete", methods=["POST"])
def passkey_login_complete():
    credential = data.get("credential")
    credential_id = credential.get("id")

    # LOOKUP CREDENTIAL FIRST (searches ALL users!)
    stored_cred = get_credential_by_id(credential_id)

    # Get user from credential lookup result
    user = get_user_by_id(stored_cred["user_id"])

    # Get GLOBAL challenge
    stored_challenge = get_global_webauthn_challenge()

    # Parse with webauthn helper (no Pydantic issues!)
    parsed_credential = parse_authentication_credential_json(json)

    # Verify and login
    verification = verify_authentication_response(
        credential=parsed_credential, ...)
```

KEY CHANGES: No email required | Credential-first lookup | Global challenge | webauthn helper

FRONTEND CHANGE (LoginScreen.tsx)

The frontend also needed a critical fix for react-native-passkey:

BEFORE (Wrong - Converted to bytes)

```
const publicKey = {
    challenge: base64urlToUint8Array(rawOptions.challenge),
    // WRONG! Causes malformed challenge in clientDataJSON
};
```

AFTER (Correct - Keep as string)

```
const publicKey = {
    challenge: rawOptions.challenge,  // Keep as base64url string!
    // react-native-passkey expects STRING, not Uint8Array
};
```

react-native-passkey expects challenge as base64url STRING, not Uint8Array

SUMMARY OF ALL CHANGES

1. New Import:

```
parse_authentication_credential_json from webauthn.helpers
```

2. New Functions:

- store_global_webauthn_challenge()
- get_global_webauthn_challenge()
- clear_global_webauthn_challenge()
- get_credential_by_id()
- get_user_by_id()

3. Modified Endpoints:

- /api/passkey/login/begin - No email, no allowCredentials
- /api/passkey/login/complete - Credential-first lookup

4. Frontend Fix:

- Pass challenge as base64url string to react-native-passkey

RESULT: Google Password Manager now shows passkey selector like Snapchat!

Database Schema (Unchanged)

The passkey_credentials JSON column in slay_users remains the same:

```
[
  {
    "credential_id": "ag-DMtYQpXyegUBeRTLIPA",
    "public_key": "pQECAyYgASFYI...",
    "sign_count": 0,
    "device_name": "android",
    "created_at": "2026-01-30T05:59:02.112173"
  }
]
```